Universidad de Valladolid

Escuela de Ingenierías Industriales

Máster Ingeniero en Modelización Matemática y Computación

Adecuación de técnicas de restauración avanzada de imágenes para dispositivos móviles

Alumno: Samuel Martinez Orna

Tutor: Santiago Aja Fernández

TÍTULO: *Adecuación de técnicas de restauración avanzada de imágenes para dispositivos móviles.*

AUTOR: Samuel Martinez Orna.

TUTOR: Santiago Aja Fernández.

DEPARTAMENTO: Teoría de la Señal y Comunicaciones e Ingeniería Telemática.

Este trabajo fue depositado en el Negociado de la Escuela de Ingenierías Industriales en la Sede del Paseo del Cauce.

En Valladolid, a

Miembros del Tribunal:

PRESIDENTE:

VOCAL:

SECRETARIO:

FECHA DE LECTURA:

CALIFICACIÓN:

A la memoria de
Óscar Pascual Garcés.

Dedicado a
Virginia Rodríguez Aragonés
por su apoyo y comprensión durante el desarrollo de este trabajo

Resumen

En la actualidad, los dispositivos móviles han sufrido una gran transformación. Se ha pasado en un tiempo muy corto de poseer un aparato con el cual únicamente se podían realizar llamadas a tener pequeños ordenadores con una capacidad de cómputo inimaginable no hace muchos años atrás. La transformación de estas máquinas ha dado lugar a que en la actualidad se puedan realizar a traves de ella gran cantidad de tareas relacionadas mayormente con el ocio y el trabajo como pueden ser la visualización de videos o la lectura y envío de correos electrónicos mediante aplicaciones especialmente diseñadas.

Las aplicaciones móviles por los motivos anteriores son extremadamente utilizadas debido a la penetracion de los smartphones (teléfonos inteligentes) en nuestra sociedad. Es por ello por lo que se ha decido realizar una aplicación de procesado de imágenes digitales en estos aparatos que trate de solventar algunas de las carencias que se pueden ecntrar dentro del "ecosistema" actual que a estos teléfonos soportan.

En cuanto a imágenes se refiere, las carencias mas graves que se han detectado en estos dispositivos son:

- Ruido en forma de pequeños puntos con una tonalidad que esta fuera de lugar y no se corresponde con la realidad.
- Elementos indeseados dentro de las imagenes, texto superpuesto o marcas de deterioro de la imagen.

Para solventar cada una de ellas se ha propuesto un enfoque diferente:

- Filtrado de difusion anisótropa.
- Inpainting o repintado de imágenes.

Índice general

1. Introducción **15**
- 1.1. Objetivos . . . 17
- 1.2. Plan de trabajo . . . 18
- 1.3. Estructura de la memoria . . . 18

2. Estado del arte **21**
- 2.1. Teoría de la difusión anisótropa . . . 21
  - 2.1.1. Evolución . . . 21
  - 2.1.2. Principales Papers . . . 23
- 2.2. Técnicas de *Inpainting* . . . 29
  - 2.2.1. Definición . . . 29
  - 2.2.2. Origen . . . 29
  - 2.2.3. Aplicación . . . 30

3. Métodos seleccionados **49**
- 3.1. Filtrado de difusión anisótropa . . . 49
  - 3.1.1. Modelo explícito . . . 50
  - 3.1.2. Modelo semi-implícito . . . 53
- 3.2. *Inpainting* . . . 57
  - 3.2.1. Contexto del método . . . 58
  - 3.2.2. Método . . . 58

4. Discretización numérica **65**
- 4.1. Filtrado de difusión anisótropa . . . 65
  - 4.1.1. Filtro explícito . . . 66
  - 4.1.2. Filtro semi-implícito . . . 72
  - 4.1.3. Imágenes a color . . . 78
- 4.2. *Inpainting* . . . 79
  - 4.2.1. Esquema discreto . . . 79
  - 4.2.2. Difusión Anisótropa para *Inpainting* . . . 82
  - 4.2.3. Detalles de implementación . . . 84

5. Implementación **87**
- 5.1. Análisis numérico . . . 87
  - 5.1.1. *Scilab* . . . 87
  - 5.1.2. *MATLAB* . . . 88
  - 5.1.3. *GNU Octave* . . . 88
  - 5.1.4. Elección . . . 88

5.2. Plataformas móviles . . . 89
5.2.1. iOS . . . 90
5.2.2. Android . . . 91
5.2.3. Elección de la plataforma . . . 91
5.3. Herramientas para el entorno móvil . . . 92
5.3.1. Android SDK . . . 92
5.3.2. Android NDK . . . 93
5.3.3. Eclipse . . . 93
5.3.4. Cygwin . . . 93

6. Experimentos y resultados obtenidos **95**
6.1. Filtrado de difusión anisótropa . . . 95
6.1.1. Influencia del coeficiente de difusividad . . . 96
6.1.2. Influencia de la elección del paso temporal . . . 97
6.1.3. Influencia del número de iteraciones . . . 98
6.1.4. Resultados con otras imágenes . . . 99
6.1.5. Comparativa del coste temporal . . . 100
6.2. *Inpainting* . . . 101
6.2.1. Elección de la dirección de propagación de los isófotos . . . 101
6.2.2. Uso de la aproximación rápida . . . 103
6.2.3. Comparativa del algoritmo *MATLAB* con la aplicación *Android* . . . 104
6.3. Conclusiones . . . 107

7. Comparativa con otras aplicaciones *Android* **109**
7.1. Aplicaciones de retoque o edición de fotografías . . . 109
7.1.1. Photoshop Express . . . 110
7.1.2. Camera 360 Ultimate . . . 111
7.1.3. PicsArt - Estudio . . . 112
7.1.4. Pixlr-o-matic . . . 113
7.1.5. Photo Editor . . . 115
7.1.6. Otras aplicaciones . . . 116
7.2. Conclusiones . . . 118

8. Conclusiones, aportaciones y líneas futuras **121**
8.1. Conclusiones . . . 121
8.1.1. Filtrado de difusión anisótropa . . . 121
8.1.2. *Inpainting* . . . 122
8.1.3. Programación en *Android* . . . 122
8.2. Aportaciones . . . 123
8.3. Líneas futuras . . . 123

A. Manual de usuario de la aplicación *PhotoRestore* **129**
A.1. Instalación . . . 129
A.1.1. Dispositivo físico *Android* . . . 129
A.1.2. Emulador *Android Virtual Device* . . . 130
A.2. Uso de la aplicación . . . 132
A.2.1. Pantalla principal . . . 132
A.2.2. Explorador de ficheros . . . 133
A.2.3. Menú de imagen . . . 134

A.2.4. *Inpainting* . . . 136
A.2.5. Reducción de ruido . . . 138

B. Especificaciones técnicas **141**
B.1. Ordenador portátil . . . 141
B.2. Tablet . . . 141

C. Implementación Matlab **143**
C.1. Filtrado de difusion anisótropa . . . 143
C.1.1. Filtro de Perona-Malik . . . 143
C.1.2. Filtro de Weickert . . . 147
C.2. Inpainting . . . 155

D. Aplicación Android **165**
D.1. Código C . . . 165
D.2. Código Java . . . 191
D.3. Mapeadores de la vista . . . 248
D.4. Mapeadores de los menús . . . 257
D.5. Contenido de los campos estáticos . . . 259
D.6. Configuración de la aplicación . . . 262

Capítulo 1

Introducción

Durante los últimos años, los teléfonos móviles han sufrido una grandrísima evolución. Se ha pasado de poseer un dispositivo con el que únicamente se podían hacer llamadas y enviar mensajes de texto a máquinas con una gran capacidad de cómputo con las cuales se puede navegar por Internet, escuchar música, visionar películas en alta definición, enviar correos electrónicos, ...

Cuando se comenzaron a comercializar entre el gran público las características principales que poseía un teléfono móvil eran una pequeña pantalla realizada en base a una matriz de puntos, un sistema cerrado que incluía las funciones básicas (agenda, despertador, realización de llamadas, envío de mensajes, ...) y una electrónica que cumpliese con el cometido de una forma sencilla. Un claro ejemplo de este tipo de teléfonos es el que se muestra en la 1.1.
El Nokia 3310 fue uno de los teléfonos móviles más populares de su tiempo.

Figura 1.1: Nokia 3310

Desde hace algún tiempo, los fabricantes han pasado a comercializar dispositivos muchísimo más avanzados y, en su momento, pasaron a denominarlos *smartphones* (teléfonos inteligentes). Los priméros *smartphones* comenzaron a fabricarlos empresas como pueden ser RIM o Palm, enfocándolos a la productividad empresarial. Eran dispositivos que estaban orientados a la gestión de agenda de reuniones y de cuentas de correos electrónicos, integraban en un mismo dispositivo una PDA y un teléfono móvil.

La aparición de *smartphones* para el consumo comercial hace unos cinco años ha revolucionado la concepción que se tenía hasta ahora del teléfono móvil. Esta revolución se inició con dispositivos como son el *iPhone* de *Apple*, en 2007, y del *HTC Dream* como primer referente del "universo" *Android*, en 2008. La mayoría del mercado de telefonía móvil, a dia de hoy, está copado los *smartphones*.

Estos teléfonos tienen una gran capacidad de cómputo comparados con sus antecesores debido a que en su interior alojan procesadores que disponen de varios núcleos con velocidades que ya superan el Gigahercio y su memoria RAM ha ido creciendo poco a poco hasta estar llegar al Gigabyte de almacenamiento. Las pantallas también han cambiado, siendo ahora pantallas multitáctiles a color con grandes dimensiones. El tamaño de estas pantallas se mueve entre las tres y las cinco pulgadas. Actualmente estos teléfonos también disponen de buenas cámaras fotográficas.

El software que atesoran en su interior cada dia va pareciéndose más al de un ordenador. Las funcionalidades que en la actualidad poseen los *smartphones* son muy amplias. Estas funcionalidades van desde las funcionalidades más básicas de un teléfono móvil a la gestión de correos electrónicos, la visualización de películas en alta definición, la lectura de libros y los juegos adaptados para este tipo de dispositivos. Hoy en dia prácticamente se han convertido los teléfonos móviles en las nuevas videoconsolas portátiles.

Figura 1.2: Samsung Galaxy Nexus

También se ha introducido recientemente el concepto de *tablet*. Estos surgieron inicialmente con el *iPad* en 2010 al que siguieron otros dispositivos posteriormente. Los *tablets* se caracterizan por tener una capacidad de procesamiento muy similar a la de los *smartphones*, pero con unas pantallas mucho más grandes. Las pantallas de estos dispositivos suelen tener tamaños que oscilan entre las siete y las diez pulgadas.

Los dos ejemplos más representativos hoy en dia de este tipo de dispositivos se muestran el la figura 1.3. El *iPad* es el dispotivo de *Apple*, mientras que *Asus Nexus 7* es la última creación dentro del universo *Android*.

Debido a esta gran evolución en características y que estos dispositivos pueden soportar algortimos que antes eran impensables se ha decido a realizar este trabajo.

Generalmente los usuarios de este tipo de dispositivos pueden tomar fotografías con las

Figura 1.3: *Apple iPad* y *Asus Nexus 7*

cámaras que se incorporan. A muchos de los usuarios les gustaría retocar alguna de estas fotografías ya que en ellas aparece ruido y otros tipos de elementos que son indeseables. Es por ello por lo que se quieren aportar con esta aplicación dos tipos de funcionalidades novedosas:

- Eliminación de ruido de las imágenes mediante el *filtrado de difusión anisótropa*.
- Eliminación de objetos indeseados en las imágenes como pueden ser textos superpuestos o marcas de deterioro mediante *Inpainting de imágenes*.

1.1. Objetivos

Este proyecto está orientado a estudiar los métodos de *filtrado de difusión anisótropa* y de *inpainting* para crear con ellos una aplicación que pueda ejecutarse en dispositivos móviles y que trate de paliar la falta de aplicaciones que comprendan técnicas de procesado avanzado de imágenes.
Por lo tanto, se establecen los siguientes objetivos:

- Estudiar las técnicas pertenecientes al campo del *filtrado de difusión anisótropa* para realizar eliminación de ruido en imágenes. Tras haber realizado un estudio de las técnicas existentes se realizará la elección de una de ellas para su implementación.
- Estudiar las diferentes técnicas y enfoques de *inpainting* para tratar de realizar restauraciones de zonas dañadas u ocluídas en imágenes digitales.
- Implementación de al menos una de las técnicas de *filtrado de difusión anisótropa* en un software de análisis numérico para conocer su comportamiento y analizar los puntos críticos donde optimizar su implementación.
- Implementación de al menos una de las técnicas de *inpainting* mediante un software de análisis numérico para comprender su comportamiento y analizar las regiones críticas del código donde mejorar y optimizar su implementación.

- Estudiar las plataformas para dispositivos móviles existentes con el objetivo de seleccionar la que cumpla una serie de criterios y requisitos mínimos. También se debe tener en cuenta que la plataforma elegida debe dar facilidades al autor para trabajar sobre ella.
- Adaptación de las técnicas anteriormente programadas en software numérico a un lenguaje de programación que pueda ejecutar después de la plataforma móvil elegida.
- Creación de una aplicación para una de las plataformas de los dipositivos móviles existentes en la actualidad donde se puedan ejecutar los algoritmos de las técnicas propuestas.
- Validación de los resultados obtenidos para elegir y justificar los parámetros y ciertas decisiones tomadas en los cálculos y su implementación.

1.2. Plan de trabajo

Para cumplir los objetivos anteriores se ha planteado un plan de trabajo mediante el cual se espera poder obtener los mejores resultados posibles. Cada parte del desarrollo se va a centrar en un área concreta tras tener un conocimiento de toda la base anterior necesaria.
De esta forma, se realizarán los diseños un trabajo por fases:

Primera fase: En este apartado se estudiará el estado del arte de los campos que se va a proceder a estudiar. Por un lado se estudiarán los métodos del *filtrado de difusión anisótropa* y por otro se estudiarán los existentes dentro del *inpainting*.

Segunda fase: Tras realizar un análisis de los métodos se seleccionarán los que se vayan a emplear. Los métodos seleccionados para cada modalidad de restauración se implementarán mediante un software de análisis numérico para comprender su funcionamiento y observar si se pueden implementar mejoras.

Tercera fase: Como último punto del trabajo y tras haber estudiado a fondo (incluyendo la implementación) los métodos, se procederá a crear la aplicación para una plataforma de dispositivos móviles.

1.3. Estructura de la memoria

El presente trabajo puede dividirse en cuatro partes bien diferenciadas. En la primera de las partes que comprende los capítulos dos, tres y cuatro se exponen los métodos matemáticos y sus discretizaciones. Dentro del capítulo dos se expone el estado del arte relativo al *filtrado de difusión anisótropa* y a los enfoques de *inpainting*. El tercer capítulo se centra en los métodos elegidos de cada técnica, exponiendo sus bases matemáticas. Y

En la segunda parte, que concide con el quinto capítulo, se exponen los motivos de la elección sobre el software y las plataformas de desarrollo elegidas para desarrollar el trabajo.

Los capítulos sexto y séptimo, que conforman la tercera parte, están dedicados a la exposición de los experimentos realizados y los resultados obtenidos, el primero de ellos. Dentro de este capítulo se exponen los motivos que han llevado al algoritmo que finalmente se ha implementado.
Mientras que el capítulo séptimo se trata como un cierre del trabajo donde se exponen las conclusiones obtenidas con él, las aportaciones realizadas y también las posibles formas de contribuir en el futuro a esta línea de trabajo.

Finalmente, tras el grueso del trabajo se incorporan varios apéndices. Estos apéndices incorporan diversa información. El primero de los apéndices el un manual de usuario para la aplicación que se ha implementado. El segundo, incorpora la información sobre las principales especificaciones técnicas de los dispotivos utilizados para la realización del este trabajo. La tercera parte de este método incorpora todo el código escrito en *MATLAB*. La cuarta y última parte, supone todo el código desarrollado para implementar la aplicación para el dispositivo móvil. Dentro de esta última parte se incluye la adaptación de los algoritmos de restauración de imágenes, los ficheros de configuración de la aplicación, los mapeadores de la vista y también los del controlador.

Capítulo 2

Estado del arte

En el capítulo anterior se han nombrado los problemas que se quieren abordar con este trabajo. Dichos problemas son parte del conjunto de técnicas incluídas en el campo del procesado de imagen.

El principal objetivo de este capítulo es realizar una presentación e introducción al estado del arte actual en las técnicas que se van a utilizar, por lo que se dividirá en dos partes diferentes.

Inicialmente se introducirá el *filtrado de difusión anisótropa* donde se explicará su motivación y origen. Además, también se expondrá su base matemática y los principales artículos propuestos en el terreno mediante una breve explicación.

En la segunda sección de este capítulo se definirá qué es el *inpainting*, debido a que es una palabra inglesa cuya traducción podría acercarse a "restauración". Además, también se mostrarán los enfoques más conocidos de los métodos de *inpainting*, para ello se realizará un pequeño resumen de cada uno de ellos en base a los artículos más destacados aparecidos hasta el momento.

2.1. Teoría de la difusión anisótropa

Se habla de técnicas de filtrado basadas en la difusión cuando las operaciones de limpieza de ruido, suavizado y realce de bordes se modelan como un proceso de difusión entre celdas adyacentes, de manera que con el paso del tiempo las zonas similares se vuelven más homogéneas y los bordes se realzan.
La difusión será denominada como *isótropa* en el caso en que la difusión se produzca por igual en todas las direcciones.
Si, por el contrario, la difusión varía dependiendo de la dirección, entonces se habla de *anisótropa*.

2.1.1. Evolución

En los últimos años se han definido diversos modelos para el suavizado y mejora de bordes de las imágenes basados tanto en procesos de difusión lineal como no lineal. Estos

intentos surgen de la idea de analizar las imágenes a diferentes niveles de resolución. El primer modelo fue propuesto por Witkin [42] y posteriormente fue más desarrollado por Koenderink [43]. En este modelo la imagen resultante se obtenía a partir de la convolución de la imagen original con un *kernel gaussiano* de una cierta varianza σ_t:

$$I(x, y, t) = I_0(x, y) * G(x, y; \sigma_t) \tag{2.1}$$

Cambiando la varianza σ_t, se obtiene un conjunto de imágenes a distintas resoluciones (véase la Figura 2.1).

El principal problema de esta técnica, es que las imágenes resultantes tenían normalmente unas resoluciones muy bastas, que impedían determinar exactamente la localización de los bordes de las estructuras.
Además, los bordes que se veían en estas resoluciones habían modificado sus posiciones respecto a las originales.

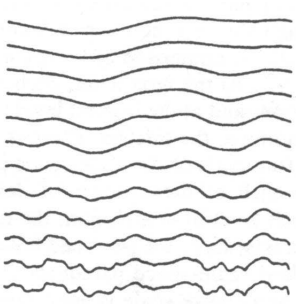


Figura 2.1: Representación de las señales obtenidas mediante la convolución de la original (señal de más abajo) con un *kernel gaussiano*, cuya varianza se va incrementando (desde abajo hacia arriba).

Koenderink en [43] apuntó que esta serie de imágenes obtenidas derivadas de un sólo parámetro se podían ver como la solución de la ecuación de difusión del calor:

$$I_t = \nabla I = (I_{xx} + I_{yy}) \tag{2.2}$$

con la condición inicial $I(x, y, 0) = I_0(x, y)$, la imagen original. Koenderink formuló la ecuación de difusión debido al planteamiento de dos criterios:

1. Causalidad: Al disminuir la resolución no pueden generarse colores espúreos en la imagen.

2. Homogeneidad e isotropía: La borrosidad debe ser invariante en el espacio.

Estos dos criterios llevan a la formulación de la ecuación del calor, pero el segundo criterio se puede sustituir por algo más útil. Esto fue lo que hicieron P. Perona y J. Malik [1], consiguiendo así desarrollar un modelo de suavizado multiescala y mejora de bordes, que ha supuesto una herramienta muy útil para la limpieza de ruido en la imagen.
Perona y Malik [1] enunciaron los criterios que se debían cumplir para obtener un buen modelo de descripción de imágenes multiescala. Estos criterios son:

1. Causalidad: Idéntica a la que estableció Koenderink.

2. Localización inmediata: En cada resolución los límites de las regiones deben ser distinguidos y deben coincidir con los originales.

3. Suavizado según zona: El suavizado debe ser mayor dentro de una región que con las regiones vecinas.

2.1.2. Principales Papers

2.1.2.1. Perona Y Malik

Scale-space and edge detection using anisotropic diffusion, (IEEE Trans. Pattern Anal. Mach. Intell.), Julio 1990. [1]

Abstract

A new definition of scale-space is suggested, and a class of algorithms used to realize a diffusion process is introduced. The diffusion coefficient is chosen to vary spatially in such a way as to encourage intraregion smoothing rather than interregion smoothing. It is shown that the 'no new maxima should be generated at coarse scales' property of conventional scale space is preserved. As the region boundaries in the approach remain sharp, a high-quality edge detector which successfully exploits global information is obtained. Experimental results are shown on a number of images. Parallel hardware implementations are made feasible because the algorithm involves elementary, local operations replicated over the image

Resumen

La idea básica de su trabajo es la posibilidad de convolucionar con un kernel Gaussiano variante con el tiempo (scale-space filtering):

$$I(x, y, t) = I_0(x, y) * G(x, y; t)$$

Esta clase de filtrado presenta ciertas desventajas, fundamentalmente una distorsión espacial, fruto del kernel gaussiano. Plantean un nuevo paradigma para generar imágenes multiescala. Para que las descripciones tengan *sentido semántico* deben cumplir:

1. *Causalidad:* No deben aparece detalles espúreos.

2. *Localización inmediata:* A cada resolución los bordes deben estar bien definidos.

3. *Suavizado basado en elementos:* Se da preferencia al suavizado intraregión frente a interregión.

Plantean un método basado en difusión anisótropa. Parten de la siguiente ecuación:

$$I_t = \text{div}(c(x, y, t)\nabla I) = c(x, y, t)\Delta I + \nabla c \cdot \nabla I$$

y el coeficiente de difusión $c(x, y, t)$ lo definen como una función decreciente del gradiente:

$$c(x, y, t) = g(\|\nabla I(x, y, t)\|)$$

Plantean una discretización de la ecuación de difusión basada en cuatro flujos:

$$I_{i,j}^{t+1} = I_{i,j}^{t} + \lambda[c_N \cdot \nabla_N I + c_S \cdot \nabla_S I + c_E \cdot \nabla_E I + c_W \cdot \nabla_W I]_{i,j}^{t}$$

donde $\nabla_N I_{i,j} = I_{i-1,j} - I_{i,j}$ y $c_{N_{i,j}}^{t} = g(||(\nabla I)_{i+(1/2),j}^{t}||)$. Para la función $g(x)$ plantea dos posibilidades:

$$g_1(x) = e^{-\left(\frac{||x||}{K}\right)^2} \tag{2.3}$$

$$g_2(x) = \frac{1}{1+\left(\frac{||x||}{K}\right)^2} \tag{2.4}$$

2.1.2.2. Gerig

Nonlinear anisotropic filtering of MRI data. (IEEE Tr. Medical Imaging). Junio 1992. [37]

Abstract

In contrast to acquisition-based noise reduction methods a postprocess based on anisotropic diffusion is proposed. Extensions of this technique support 3-D and multiecho magnetic resonance imaging (MRI), incorporating higher spatial and spectral dimensions. The procedure overcomes the major drawbacks of conventional filter methods, namely the blurring of object boundaries and the suppression of fine structural details. The simplicity of the filter algorithm permits an efficient implementation, even on small workstations. The efficient noise reduction and sharpening of object boundaries are demonstrated by applying this image processing technique to 2-D and 3-D spin echo and gradient echo MR data. The potential advantages for MRI, diagnosis, and computerized analysis are discussed in detail.

Resumen

Este artículo de Gerig es realmente la primera implementación práctica *real* de la idea planteada por Perona y Malik. Sigue la misma ecuación de difusión:

$$\frac{\partial}{\partial t} u(\vec{x}, t) = \mathrm{div}(c(\vec{x}, t), \nabla u(\vec{x}, t))$$

Para $c(\vec{x}, t), \nabla u(\vec{x}, t)$ utiliza ecuaciones muy similares a las del caso anterior. Plantea una nueva formulación discreta (ver eq. (2.5)).

$$\begin{aligned}
\frac{\partial}{\partial t} I(\vec{x}, t) &= \frac{1}{\Delta x^2}\left[c\left(x+\frac{\Delta x}{2}, y, t\right)(I(x+\Delta x, y, t) - I(x, y, t))\right. \\
&\quad \left. - \; c\left(x-\frac{\Delta x}{2}, y, t\right)(I(x, y, t) - I(x-\Delta x, y, t))\right] \\
&\quad + \frac{1}{\Delta y^2}\left[c\left(x, y+\frac{\Delta y}{2}, t\right)(I(x, y+\Delta y, t) - I(x, y, t))\right. \\
&\quad \left. - \; c\left(x, y-\frac{\Delta y}{2}, t\right)(I(x, y, t) - I(x, y-\Delta y, t))\right] \\
&= \Phi_E - \Phi_W + \Phi_N - \Phi_S
\end{aligned} \tag{2.5}$$

De manera más sencilla se puede representar como

$$\begin{aligned} I(t+\Delta t) &\approx I(t) + \Delta t \, \frac{\partial}{\partial t} \, I(t) \\ &= I(t) + \Delta t \, (\Phi_E - \Phi_W + \Phi_N - \Phi_S) \end{aligned}$$

En este paper se plantea también el estudio de algunos parámetros, como el parámetro K que aparece en las ecuaciones (2.3) y (2.4). Si σ_n^2 es la varianza de ruido en la imagen, una buena elección de K es

$$1{,}5\sigma_n < K < 2\sigma_n$$

El escalón temporal Δt debe elegirse para lograr que los flujos que afectan a la imagen no hagan esta inestable. Se elige

$$\Delta t \leq \frac{1}{1+n} \tag{2.6}$$

siendo n el número de nodos del vecindario. (Realmente se toma el número de flujos que actúan sobre el pixel).

2.1.2.3. Catté

Image selective smoothing and edge detection by nonlinear diffusion (SIAM J. Number Anal.) Feb. 1992 [34]

Abstract

A new version of the Perona and Malik theory for edge detection and image restoration is proposed. This new version keeps all the improvements of the original model and avoids its drawbacks: it porved to be stable in presence of noise, with existence and uniqueness results. Numerical experiments on natural images presented.

Resumen

Parten de la ecuación de difusión

$$\frac{\partial}{\partial t} \, u = \operatorname{div}(g(|\nabla u|)\nabla u), \;\; u(0) = u_0$$

Realizan una serie de críticas al modelo de Perona y Malik y son las siguientes:

- En caso de tener una imagen ruidosa, este ruido introduce variaciones no acotadas del gradiente ∇u. El propio modelo no puede con este ruido.
- Las funciones $g(s)$ usadas por Perona y Malik en [1] no son correctas para la ecuación de difusión. Para obtener una solución de la ecuación y que ésta sea única, debe cumplirse la condición de que $sg(s)$ sea no decreciente. En la práctica esto significa que imágenes muy parecidas pueden producir soluciones divergentes, y por lo tanto diferentes bordes.
- Se trata de un problema "ill-posed" o mal puesto.

Para su nueva aproximación parten de

$$\frac{\partial}{\partial t}u = \text{div}(g(|D\,G_\sigma * u|)\nabla u),\ u(0) = u_0$$

La diferencia con el modelo de Perona y Malik es que reemplazan el gradiente (∇u) dentro de la función g por su estimador $D\,G_\sigma * u$. De este modo no es necesario realizar un suavizado previo para eliminar ruido y se puede probar que la solución es única. (En la práctica, este estimador se puede reemplazar por un filtrado pasobajo realizado por un *kernel Gaussiano*).

Con todo ello, plantean un esquema iterativo que converge a la solución

$$\frac{du^{n+1}}{dt}(t) = \text{div}(g(|\nabla G_\sigma * u^n(t)|)\nabla u^{n+1}(t))$$

que lleva a una nueva discretización del problema de la difusión, concretamente a

$$\frac{u^{n+1} - u^n}{\Delta t} + \mathbf{A}_h(u^n)u^{n+1} = 0,\ n \geq 0$$

donde $\mathbf{A}_h$ es una matriz tridiagonal por bloques y definida positiva.
La matriz total $\mathbf{I} + \Delta t\mathbf{A}_h(u^n)$ es invertible.

2.1.2.4. Weickert

Efficient and reliable schemes for nonlinear diffusion filtering, (IEEE Tr. Image Proc) Mar. 1998. [3].

Abstract

Nonlinear diffusion filtering is usually performed with explicit schemes. They are only stable for very small time steps, which leads to poor efficiency and limits their practical use. Based on a recent discrete nonlinear diffusion scale-space framework we present semi-implicit schemes which are stable for all time steps. These novel schemes use an additive operator splitting (AOS), which guarantees equal treatment of all coordinate axes. They can be implemented easily in arbitrary dimensions, have good rotational invariance and reveal a computational complexity and memory requirement which is linear in the number of pixels. Examples demonstrate that, under typical accuracy requirements, AOS schemes are at least ten times more efficient than the widely used explicit schemes.

Resumen

El artículo presenta una una solución a la ecuación de difusión. Parte del esquema de difusión no lineal de Catté y cía. [34]. Este esquema es una discretización de la ecuación de difusión

$$\partial_t u = \text{div}\left[g(|\nabla u_\sigma|^2)\nabla u\right]$$

siendo $u(x,t)$ la imagen a procesar. Al discretizar la ecuación para una dimensión, el esquema explícito (tras operar) queda

$$u^{k+1} = [I + \tau A(u^k)]u^k$$

Autores	**Discretización**	**Comentarios**
Catté y cía.	$u^{k+1} = [I + \tau \sum_{l=1}^{m} A_l(u^k)]u^k$	Ecuación explícita
Weickert y cía.	$u^{k+1} = [I - \tau \sum_{l=1}^{m} A_l(u^k)]^{-1}u^k$ $u^{k+1} = \frac{1}{m}\sum_{l=1}^{m}[I - m\tau A_l(u^k)]^{-1}u^k$	Ecuación semi-implícita AOS
Perona y Malik	$I_s^{t+1} = I_s^t + \frac{\lambda}{\lvert\eta_s\rvert}\sum_{p\in\eta_s} g(\nabla I_{s,p})\nabla I_{s,p}$	

Cuadro 2.1: Comparativa entre Discretizaciones

Los autores proponen un esquema pseudo implícito que da lugar a la ecuación

$$u^{k+1} = [I - \tau A(u^k)]^{-1}u^k$$

Para dimensión m el esquema explícito será

$$u^{k+1} = [I + \tau \sum_{l=1}^{m} A_l(u^k)]u^k$$

y el implícito

$$u^{k+1} = [I - \tau \sum_{l=1}^{m} A_l(u^k)]^{-1}u^k$$

Al ser muy complicado de calcular para $m \geq 2$, en se opta por un esquema AOS (Additive Operatot Splitting):

$$u^{k+1} = \frac{1}{m}\sum_{l=1}^{m}[I - m\tau A_l(u^k)]^{-1}u^k$$

La ventaja de este esquema es que τ (el escalón temporal) no está sujeto a la restricción $\tau \leq 1/2m$, ya que la ecuación va a ser siempre estable.

2.1.2.5. Fischl

Learning an integral equation approximation to nonlinear anisotropic diffusion in image processing, (IEEE Trans. Pattern Anal. Mach. Intell.), Abril 1997. [35]

Abstract

Multiscale image enhancement and representation is an important part of biological and machine early vision systems. The process of constructing this representation must be both rapid and insensitive to noise, while retaining image structure at all scales. This is a complex task as small scale structure is difficult to distinguish from noise, while larger scale structure requires more computational effort. In both cases, good localization can be problematic. Errors can also arise when conflicting results at different scales require cross-scale arbitration. Structure sensitive multiscale techniques attempt to analyze an image at a variety of scales within a single image. Various techniques are compared. In this paper, we present a technique which obtains an approximate solution to the partial differential

equation (PDE) for a specific time, via the solution of an integral equation which is the nonlinear analog of convolution. The kernel function of the integral equation plays the same role that a Green's function does for a linear PDE, allowing the direct solution of the nonlinear PDE for a specific time without requiring integration through intermediate times. We then use a learning technique to approximate the kernel function for arbitrary input images. The result is an improvement in speed and noise-sensitivity, as well as providing a means to parallelize an otherwise serial algorithm

2.1.2.6. Black 1998

Robust anisotropic diffusion, (IEEE Tr. Imag. Proc). Marzo 1998 [41]

Abstract

Relations between anisotropic diffusion and robust statistics are described in this paper. Specifically, we show that anisotropic diffusion can be seen as a robust estimation procedure that estimates a piecewise smooth image from a noisy input image. The "edge-stopping" function in the anisotropic diffusion equation is closely related to the error norm and influence function in the robust estimation framework. This connection leads to a new "edge-stopping" function based on Tukey's biweight robust estimator that preserves sharper boundaries than previous formulations and improves the automatic stopping of the diffusion. The robust statistical interpretation also provides a means for detecting the boundaries (edges) between the piecewise smooth regions in an image that has been smoothed with anisotropic diffusion. Additionally, we derive a relationship between anisotropic diffusion and regularization with line processes. Adding constraints on the spatial organization of the line processes allows us to develop new anisotropic diffusion equations that result in a qualitative improvement in the continuity of edges

2.1.2.7. Black 1999

Edges as outliers: anisotropic smoothing using local image statistics, (LNCS) Septiembre 1999. [40]

Abstract

Edges are viewed as statistical outliers with respect to local image gradient magnitudes. Within local image regions we compute a robust statistical measure of the gradient variation and use this in an anisotropic difusion framework to determine a spatially varying "edge- stopping" parameter σ. We show how to determine this parameter for two edge-stopping functions described in the literature (Perona-Malik and the Tukey biweight). Smoothing of the image is related the local texture and in regions of low texture, small gradient values may be treated as edges whereas in regions of high texture, large gradient magnitudes are necessary before an edge is preserved. Intuitively these results have similarities with human perceptual phenomena such as masking and "popout". Results are shown on a variety of standard images.

2.2. Técnicas de *Inpainting*

La palabra *inpainting* es un término procedente del mundo anglosajón. Si se tratase de traducir de forma literal sería algo así como "pintar dentro" o, como prefiero traducirla "repintar".
Dentro de este término se engloban en el campo del procesado de imagen técnicas variadas para la reconstrucción de regiones mediante información existente en la imagen.

Dado que el término es complejo y los métodos abarcan una buena cantidad de enfoques diferentes se va a tratar de exponer el estado actual de la técnica en este apartado.
En esta exposición se encontrará inicialmente una definición mediante la cual se podrá hacer una idea de qué es lo que pretende este tipo de métodos.
Seguidamente, se introducirá al lector en el origen y la evolución histórica del método.
Y, finalmente, se expondrán los diferentes métodos más destacados en los diferentes enfoques y se mostrarán algunos de sus resultados.

2.2.1. Definición

Dada una imagen y una región Ω en su interior, el problema consiste en modificar los valores de los píxeles pertenecientes a Ω para que los píxeles de la región no destaquen de los que la rodean. El propósito del método debe ser restaurar porciones de una imagen (por ejemplo, una fotografía vieja donde los pliegues y arañazos han dejado vacíos en la imagen) o eliminar elementos indeseados presentes en la imagen (por ejemplo, un micrófono que aparece en un *frame* o fotograma de una película).

La región Ω siempre es dada por el usuario, por lo que la localización de Ω no es parte del problema de *inpainting*. Casi todos los algoritmos de *inpainting* tratan Ω como una restricción fuerte, mientras que otros métodos permiten una relajación de las fronteras de Ω.

Esta definición, dada para un problema de una sola imagen, puede extenderse de forma natural a un caso de imágenes múltiples y puede cubrir tanto el problema para imagen como para vídeo. Sin embargo, en este estudio no se considera la restauración de vídeos al igual que tampoco se considera la de superficies (por ejemplo, rellenar agujeros en escáneres en tres dimensiones).

2.2.2. Origen

El término *inpainting* fue dado en la restauración de arte, donde también se denomina *retoque*. El arte medieval comenzó a restaurarse en la época del Renacimiento. Los motivos principales para realizarlo eran los de actualizar las imágenes medievales y los de rellenar espacios. La necesidad de retocar la imagen de una forma discreta y no intrusiva extendió estos métodos de las pinturas a la fotografía y los vídeos. Los propósitos eran los mismos: eliminar el deterioro (por ejemplo, arañazos y manchas de polvo en las películas de cine) o añadir o quitar elementos (por ejemplo, la eliminación de enemigos políticos en tiempos de Stalin en las fotos de la U.R.S.S.). Se puede observar un ejemplo de algo similar en la Figura 2.2.

En el dominio digital, el problema de *inpainting* apareció inicialmente bajo el nombre

Figura 2.2: El problema de inpainting. De izquierda a derecha: Imagen original, Máscara de *inpainting* Ω en negro y Resultado de inpainting de [31]

de "ocultación del error" en telecomunicaciones, debido a la necesidad era rellenar bloques en la imagen que habían sido perdidos durante la transmisión de los datos. Uno de los primeros trabajos que se pueden denominar dentro de esta técnica fue llamado "desoclusión de la imagen", puesto que trataba el "agujero" de la imagen como un objeto que ocultaba algo y debía ser eliminado y la imagen existente en el fondo deberia ser el resultado de la restauración. Se han utilizado términos muy populares para denotar los algoritmos de *inpainting* como pueden ser "image completion" (completado de la imagen) o "image fill-in" (rellenado de la imagen).

2.2.3. Aplicación

La literatura acerca de la restauración digital mediante *inpainting* puede ser agrupada en tres categorías de métodos:

- Basada en parches.
- *Sparse* o dispersión.
- Ecuaciones Derivadas Parciales (EDPs)/Métodos variacionales.

2.2.3.1. De la síntesis de texturas a la restauración basada en parches

Efros y Leung en [13] propusieron un metodo que, aunque inicialmente estuvo pensado para síntesis de texturas, se ha mostrado mas efectivo para resolver el problema de *inpainting*. El agujero en la imagen es rellenado de forma recursiva, desde la frontera del agujero hacia su interior: cada píxel "vacío" P en la frontera $\partial\Omega$ es rellenado con el valor del píxel Q (que se encuentra fuera del agujero y tiene información válida) tal que el vecindario de Q, $\Psi(Q)$ que sera una región cuadrada centrada en Q, es lo mas similar al vecindad del píxel a rellenar $\Psi(Q)$. Formalmente, esto puede ser expresado como un problema de optimización:

$$\text{Output}(P) = \text{Value}(Q), P \in \Omega, Q \notin \Omega, Q = \arg\min d(\Psi(P), \Psi(Q))$$

donde $d(\Psi(P), \Psi(Q))$ es la suma de las diferencias al cuadrado entre los vecindarios $\Psi(P)$ y $\Psi(Q)$ (considerando únicamente los píxeles disponibles):

$$d(\Psi_1, \Psi_2) = \sum_i \sum_j |\Psi_1(i,j) - \Psi_2(i,j)|^2$$

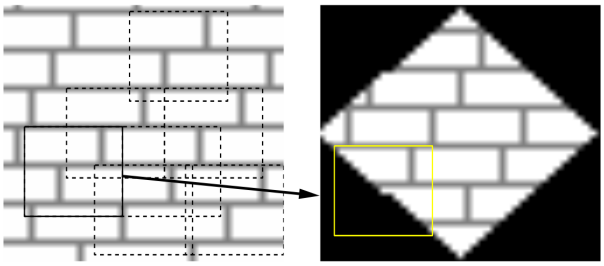


Figura 2.3: Perspectiva general del algoritmo de Efros y Leung (tomada de [13]). Dada una muestra de la textura de la imagen (izquierda), se sintetiza píxel a píxel (derecha). Para sintetizar un píxel el algoritmo busca todos los vecindarios en la imagen de muestra (cajas en la imagen de la izquierda) que son similares a los píxeles del vecindario (caja en la imagen de la derecha) y entonces elige de forma aleatoria un vecindario y toma su valor central para que sea el del nuevo píxel.

y los índices i, j recorren los píxeles que componen cada uno de los vecindarios.
Una vez que P tiene asignado su valor (esta relleno), el algoritmo pasa a otro píxel de la frontera $\partial\Omega$ continuar rellenando. En ningun momento volverá a P, puesto que su valor ya esta establecido y en el momento en que se rellena $P \notin \partial\Omega$. Para observar una pequeña descripción gráfica del algoritmo véase la Figura 2.3 y para observar un ejemplo del resultado que se puede obtener véase 2.4.

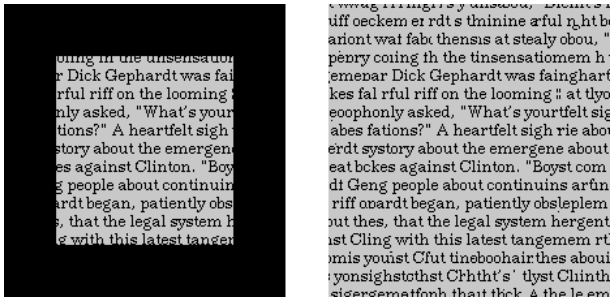


Figura 2.4: Izquierda: Imagen original donde la máscara de *inpainting* Ω se encuentra coloreada en negro. Derecha: Resultado obtenido mediante el algoritmo de *inpainting* de Efros y Leung. Imágenes obtenidas de su artículo [13].

Los principales defectos de este algotimo son su coste computacional (para cada punto P debe recorrer toda la zona de la imagen que no pertenece a Ω y encontrar el idóneo), la selección del tamaño del tamaño del vecindario Ψ (en el *paper* original es un parámetro global seleccionado por el usuario pero debería cambiar localmente dependiendo del contenido de la imagen), el orden de rellenado de los píxeles (que pueden crear fronteras inconexas para algunos objetos) y el hecho de que no se puede tratar con la perspectiva de la imagen (fue creado para sintetizar texturas frontales, aunque lso vecindarios sean siempre comparados con el mismo tamaño y orientación). Además, los resultados son probres si el agujero de la imagen es muy grande y disperso (por ejemplo, una imagen donde el ochenta por ciento de los píxeles se han perdido debido a ruido de tipo *sal y pimienta*).

Criminisi y cía. en [14] mejoraron el trabajo anterior en dos aspectos.
Primero, cambiaron el orden de relleno original por esquema en modo "onion-peel" (capas de cebolla) donde los píxeles que se encontraban en el borde de un objeto de la imagen tienen mayor prioridad que los píxeles que se encuentran en regiones planas. Esto hace posible que se restauren correctamente bordes rectos que podrían acabar desconectados con la formulación original. Un ejemplo de ello se puede contemplar en la Figura 2.5.
Segundo, en este algoritmo se copiaban parches enteros en vez de píxeles sueltos, por lo que este método es considerablemente más rápido.

Figura 2.5: Izquierda: Imagen original. Derecha: Resultado obtenido utilizando el algoritmo de Criminisi y cia. en [14]. Las imágenes han sido obtenidas de dicho artículo.

Aún así, todavía existen defectos en el algoritmo. Estos defectos son la incapacidad para tratar la perspectiva en las imágenes y la necesidad de seleccionar manualmente el tamaño del vecindario. Es más, en este caso hay que establecer dos tamaños: uno que indique el tamaño de los parches con los que comparar y otro para el parche desde el que copiar.
Además, hay que destacar que los objetos con fronteras curvas no serán restaurados correctamente.

Ashikhmin en [15] contribuyó también a mejorar el trabajo de Efros y Leung [13]. Con la idea de reducir el coste computacional del procedimiento, propuso una modificación en el modo de búsqueda del mejor candidato Q para copiar su valor al píxel P. Esta modificación se basaba en no buscar en toda la imagen sino buscar sólo entre los candidatos del vecindario de P los cuales ya habían sido restaurados. El esquema de este funcionamiento se encuentra reproducido gráficamente en la Figura 2.6.
La mejora en velocidad de cómputo que se dio con esta técnica fue considerable y también se observó un efecto de mejora en la calidad visual del resultado. Otros métodos reducen el espacio de búsqueda y el coste computacional que ello conlleva mediante la organización de los parches en tres estructuras, reduciendo la dimensión de los parches con técnicas como son Análisis de la Componente Principal (PCA) o utilizando aproximaciones aleatorias.

Mientras la mayoría de los métodos de *inpainting* tratan de ser totalmente automáticos (aparte de proveer manualmente algunos parámetros), también existen métodos en los que debe intervenir el usuario. Estos métodos dan muy buenos resultados sólo con una pequeña entrada por parte del usuario.
En el trabajo de Sun y cía. en [16], el usuario debe especificar en la región desconocida Ω las curvas correspondientes a los bordes de los objetos que se deben restaurar. Entonces la síntesis de los parches es realizada a lo largo de estas curvas dentro del agujero de la

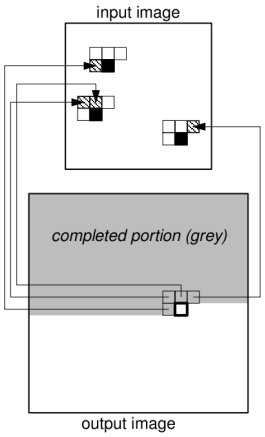


Figura 2.6: Método de sintesis de texturas de Ashikhmin (imagen sacada de [15]). Cada píxel del vecindario actual que tiene forma de L genera un píxel candidato (píxeles negros) de acuerdo a su posición original en la textura de entrada. El mejor píxel se elige sólo entre estos candidatos. Muchos píxeles en el mismo vecindario pueden generar el mismo candidato.

imagen mediante la copia de parches que se encuentran en los segmentos de estas curvas y que se encuentran en la región colindante al agujero.
Una vez completadas estas curvas, un proceso que los autores denominan *propagación de la estructura*, los píxeles vacíos restantes dentro del agujero serán rellenados mediante el uso de una técnica como la de Ashikhmin [15] con prioridades como las utilizadas por Criminisi en [14].
Barnes y cía. [17] aceleraron este método y lo hicieron interactivo. Para conseguirlo emplearon búsquedas aleatorias y combinaron en un único paso los procesos de *propagación de estructura* y síntesis de textura de Sun y cía. [16].

2.2.3.2. El papel de la dispersión

Tras la introducción de los métodos basados en parches para la síntesis de texturas de Efros y Leung [13] y la técnica de *inpainting* de Criminisi [14] ha quedado claro que los parches de una imagen proveen de un buen conjunto de candidatos, denominado diccionario, para rellenar otras partes de la imagen. La idea ha sido aplicada satisfactoriamente en otras áreas del procesado de imagen como la reducción de ruido y la segmentación.

Las representaciones más generales de tipo *sparse* que utilizan diccionarios han probado su eficiencia en el contexto de *inpainting*. Por ejemplo, mediante el uso de diccionarios sobrecompletos adaptados a la representación de la geometría y la textura de la imagen. Elad y cía. en [18] propuso una descomposición del modelo con coeficientes de tipo *sparse* para las componentes de textura y geometría de la imagen. Con ello mostró que el modelo puede ser fácilmente adaptado al *inpainting* de imágenes. A continuación se describe en mayor profundidad este modelo.
Sea u una imagen representada como un vector en $\mathbb{R}^N$. Sean las matrices D_g y D_t de tamaños $N \times k_g$ y $N \times k_t$ que representan dos diccionarios adaptados a la geometría y a la

textura de la imagen, respectivamente. Si $\alpha_g \in \mathbb{R}^{k_g}$ y $\alpha_t \in \mathbb{R}^{k_t}$ representan los coeficientes de la geometría y la textura, entonces $u = D_g\alpha_g + D_t\alpha_t$ representa la descomposición de la imagen utilizando los diccionarios almacenados en D_g y D_t. Una representación dispersa de la imagen puede ser obtenida minimizando

$$\min_{(\alpha_g,\alpha_t):u=D_g\alpha_g+D_t\alpha_t} ||\alpha_g||_p + ||\alpha_t||_p$$

donde $p = 0, 1$. Aunque el caso $p = 0$ representa la medida de dispersión (por ejemplo, el número de coordenadas que no son cero) conduce a una optimización de un problema no convexo cuya minimización es más complicada. El caso de $p = 1$ da lugar a un problema de optimización convecto y tratable conduciendo también a la dispersión.
Si se introduce una restricción mendiante una penalización (por lo tanto, una relajación en la práctica) y una regularización de la parte geométrica de la descomposición con una penalización de la seminorma de variación total, con ello Elad y cía. [18] propone el siguiente modelo variacional:

$$\min_{(\alpha_g,\alpha_t)} ||\alpha_g||_1 + ||\alpha_t||_1 + \lambda\,||u - D_g\alpha_g - D_t\alpha_t||_2^2 + \gamma\ \mathrm{TV}(D_g\alpha_g) \tag{2.7}$$

donde TV denota la variación total, $\lambda > 0$ y $\gamma > 0$.
Este modelo puede ser adaptado a un modelo para *inpainting* de imágenes. Obsérvese que $u - D_g\alpha_g - D_t\alpha_t$ puede ser interpretado como la componente del ruido de la imagen y λ es un parámetro de penalización dependiente de forma inversa del la potencia del ruido. Entonces la máscara de *inpainting* puede ser interpretada como una región donde el ruido es muy grande (infinito). Por lo tanto, si $M = 0$ identifica la región desconocida de la imagen y $M = 1$ identifica las zonas conocidas, entonces el modelo (2.7) se puede extender a la técnica de *inpainting* y reescribirlo como

$$\min_{(\alpha_g,\alpha_t)} ||\alpha_g||_1 + ||\alpha_t||_1 + \lambda\,||M(u - D_g\alpha_g - D_t\alpha_t)||_2^2 + \gamma\ \mathrm{TV}(D_g\alpha_g) \tag{2.8}$$

Si se escribe la energía en (2.8) utilizando $u_g := D_g u$ y $u_t := D_t u$ como variables desconocidas, se puede observar que $\alpha_g = D_g^+ u_g + r_g$, $\alpha_t = D_t^+ u_t + r_t$, donde D_g^+ y D_t^+ denotan las correspondientes matrices pseudoinversas y r_g y r_t se encuentran en los espacios vacíos de D_g y D_t, respectivamente.
Asumiendo por simplicidad, como en Elad y cía. [18], que $r_g = 0$ y $r_t = 0$, el modelo de (2.8) puede ser reescrito como

$$\min_{(\alpha_g,\alpha_t)} \left|\left|D_g^+ u_g\right|\right|_1 + \left|\left|D_t^+ u_t\right|\right|_1 + \lambda\,||M(u - u_g - u_t)||_2^2 + \gamma\ \mathrm{TV}(u_g) \tag{2.9}$$

Este modelo simplificado está justificado en Elad y cía. [18] por múltiples razones: es un límite superior para (2.8), es más fácil de resolver, da buenos resultados, tiene una interpretación bayesiana y es equivalenste a (2.8) si D_g y D_t son no-singulares, o se utilice la norma l^2 en lugar de la norma l^1. El modelo tiene buenas propiedades puesto que permite el uso de diccionarios adaptados para la geometría y la textura, tratando los lugares de *inpainting* como muestras desaparecidas y debido a que el modelo de dispersión tiene incluídas normas l^1 que son fáciles de resolver.

Este marco ha sido adaptado al uso de diccionarios en parches y ha sido extendido en varias direcciones como reducción de ruido en imágenes, rellenado de píxeles erróneos

(Aharon y cía [19]), reducción de ruido en imágenes a color, *demosaicing* e *inpainting* para pequeños agujeros (Mairal y cía. [20]).
Y más allá también ha sido extendida para tratar con diccionarios multiescala y para cubrir el caso de secuencias de vídeo en [21].

Para dar un pequeño repaso a este modelo se necesita más notación. Los parches de la imagen son cuadrados de tamaño $n = \sqrt{n} \times \sqrt{n}$. Sea D el diccionario de parches representado por una matriz de tamaño $n \times k$, donde los elementos del diccionario son columnas de D. Si $\alpha \in \mathbb{R}^k$ es un vector de coeficientes, entonces $D\alpha$ representa el parche obtenido mediante la combinación lineal de las columnas de D. Dada una imagen $v(i,j), i,j \in \{1,\ldots,N\}$, el propósito es encontrar un diccionario $\hat{D}$, una imagen $\hat{u}$ y coeficientes $\hat{\alpha} = \{\hat{alpha}_{i,j} \in \mathbb{R}^k : i,j \in \{1,\ldots,N\}\}$ que minimicen la energía

$$\min_{\alpha,D,u} \lambda \left|\left|v-u\right|\right|_2 + \sum_{i,j=1}^{N} \mu_{i,j} \left|\left|\alpha_{i,j}\right|\right|_0 + \sum_{i,j=1}^{N} \left|\left|D\alpha_{i,j} - R_{i,j}u\right|\right|_2 \tag{2.10}$$

donde $R_{i,j}u$ denota el parche de u centrado en (i,j) (desestimando los efectos en la frontera) y $\mu_{i,j}$ son pesos positivos.

La solución para el problema no convexo (2.10) es obtenida utilizando una minimización alternativa: un paso de codificación dispersa donde se calculan los valores $\alpha_{i,j}$ conociendo el diccionario D para todo i,j, una actualización del diccionario utilizando una secuencia de problemas de aproximación de rango uno para actualizar cada columna de D (Aharon, Elad y Bruckstein [19]) y un paso final de reconstrucción dado por la solución de

$$\min_{u} \lambda \left|\left|v-u\right|\right|_2 + \sum_{i,j=1}^{N} \left|\left|D\alpha_{i,j} - R_{i,j}u\right|\right|_2 \tag{2.11}$$

De nuevo, el problema de *inpainting* puede ser considerado como un caso de ruido no homogéneo. Definiendo para cada píxel (i,j) un coeficiente $\beta_{i,j}$ inversamente proporcional a la varianza del ruido, el valor de $\beta_{i,j} = 0$ debe ser tomado para cada píxel en la máscara de *inpainting*. Entonces el problema puede ser formulado como

$$\min_{\alpha,D,u} \lambda \left|\beta \otimes \left|v-u\right|\right|_2 + \sum_{i,j=1}^{N} \mu_{i,j} \left|\left|\alpha_{i,j}\right|\right|_0 + \sum_{i,j=1}^{N} \left|\left|(R_{i,j}\beta) \otimes (D\alpha_{i,j} - R_{i,j}u)\right|\right|_2 \tag{2.12}$$

donde $\beta = (\beta_{i,j})_{i,j=1}^{N}$ y $\otimes$ denota un producto elemento a elemento entre los dos vectores.

Con adaptaciones más adecuadas, este modelo puede ser aplicado a *inpainting* (de agujeros relativamente pequeños), de interpolación desde muestras dispersas irregulares y superresolución, a reducción de ruido en imágenes, *demosaicing* de imágenes en color y reducción de ruido e *inpainting* en vídeos. Obteniendo con ello excelentes resultados, véase [21].

2.2.3.3. EDPs y aproximaciones variacionales

Todos los métodos mencionados hasta el momento están basados en el mismo principio: una parte de una image está corrompida o sus valores no vienen dados puede ser obtenidos mediante un muestreo adecuado y copiando parches que no hayan sido corrompidos

(tomando de las regiones no corrompidas de la imagen o de un diccionario). Un punto de vista muy diferente también subyace en muchos otros artículos utilizando el principio variacional, a través de un proceso de minimización o una Ecuación en Derivadas Parciales (EDP) no necesariamente variacional.

Un primer método basado en la interpolación que se aplicaba a *inpainting* lo crearon Ogden, Adelson, Bergen y Burt [22]. En este método comienza a tratar la imagen inicial con un filtrado Gaussiano mediante convolución iterativa y submuestreo. Entonces, dado un dominio de *inpainting* Ω, se puede rellenar mediante interpolaciones lineales sucesivas diminuyendo y aumentando la resolución a diferentes niveles de la pirámide Gaussiana. La eficiencia de este enfoque esta ilustrada en la Figura 2.7.

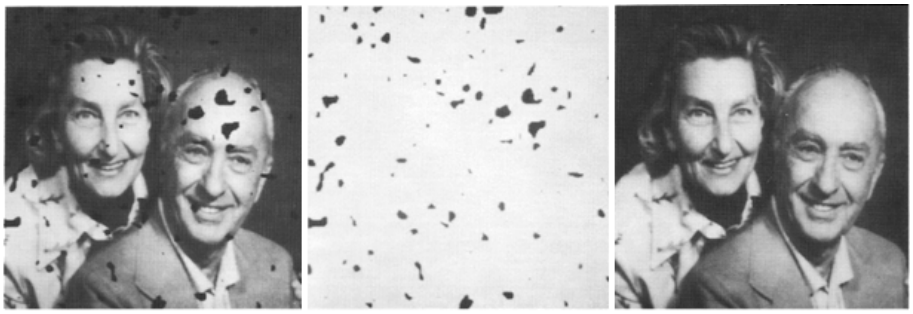


Figura 2.7: Experimento de inpainting tomado de [22]. El método utiliza una pirámide gaussiana y series de interpolaciones lineales, reducciones y aumentos de la resolución.

Masnou y Morel propusieron en [23] interpolar una imagen en escala de grises extendiendo sus isófotos (las líneas de intensidad constante) en el dominio de *inpainting* Ω. Este enfoque está mucho más próximo a las primeras aproximaciones realizadas por Kanizsa, Ullman, Horn, Mumford y Nitzberg para modelar la capacidad del sistema visual para completar bordes en una oclusión dada o dentro del contexto de una ilusión (véase la Figura 2.8).
El proceso general de rellenado contempla fenómenos conplejos que no pueden ser modelados de forma sencilla y unívoca. Sin embargo, los resultados experimentales musetran que, en situaciones de oclusiones sencillas es razonable pensar que el cerebro extrapola los bordes rotos utilizando curvas de elásticas, una longitud total menor que una máxima dada L y minimizar la energía elástica de Euler $\int |\kappa(s)|^2 \, ds$, siendo s la longitud de la curva y κ la curvatura.

El modelo de Masnou y Morel en [23] generaliza este principo a los isófotos de la imagen en escala de grises. Enunciándolo de una forma más precisa, denotando $\tilde{\Omega}$ como un dominio algo mayor que Ω, se propone en [23] extrapolar los isófotos de una imagen u, conocida en todos los puntos $(i, j) \notin \Omega$, los cuales tiene sus valores en el intervalo $[m, M]$, por una colección de curvas $\{\gamma_t\}_{t\in[m,M]}$ sin cruces recríprocos que coincidan con los isófotos de u en $\tilde{\Omega}\Omega$ y que minimiza la energía

$$\int_m^M \int_{\gamma_t} (\alpha + \beta \, |\kappa_{\gamma_t}|^p) \, ds \, dt \tag{2.13}$$

Aquí, α y β son dos parámetros dependientes del contexto. Esta energía penaliza la energía elástica de Euler, con una curvatura de potencia $p > 1$ en vez de 2, de todas las curvas de

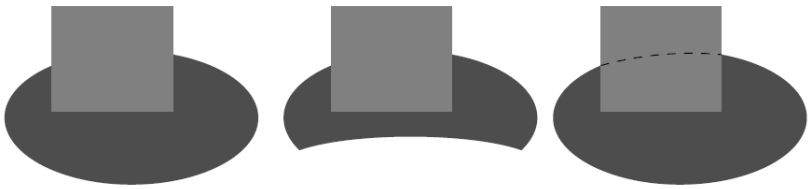


Figura 2.8: Completado amodal: el sistema visual completa automáticamente el borde tapado en la Figura de la izquierda. La Figura de enmedio muestra que aquí no se aplica ningún proceso de simetría global, partiendo de ambas Figuras se sintetiza el mismo borde. En una situación tan simple, la curva interpolada puede ser modelada como una curva elástica de Euler o, lo que es lo mismo, una curva con puntos de anclaje, tangentes en sus extremos y con oscilaciones mínimas.

extrapolación $\gamma_t, t \in [m, M]$.

Un algoritmo de *inpainting* basado en la minimización de (2.13) en el caso $p = 1$ se propuso en [23] de Masnou y Morel. Una solución minimal global es calculada utilizando una aproximación a la programación dinámica que reduce la complejidad algorítmica. El algoritmo sólo maneja dominios simples conectados, es decir, aquellos que no tienen agujeros.
Con el objetivo de conseguir hacer funcionar correctamente el algoritmo con imágenes en color, las imágenes en formato RGB son transformadas en una representación de luminancia/crominancia, como pueden ser los formatos *YCrCb* o *Lab* y cada canal es procesado de forma independiente. Este proceso esta ilustrado en la Figura 2.9.

La palabra *inpainting* fue introducida inicialmente en el contexto del procesado de imagen por Bertalmío, Sapiro, Caselles y Ballester en [6], donde se propone un modelo mediante el uso de EDP (Ecuaciones en Derivadas Parciales) que sigue el mismo espíritu que las técnicas de restauración de reales que se aplican a las obras de arte.
De una forma más precisa, se puede definir como u a la imagen en escala de grises que va a ser restaurada en Ω. A u se le aplicará un método de resolución dependiente del tiempo para la ecuación de transporte

$$\begin{aligned} u_t = \nabla^{\perp} u \cdot \nabla \Delta u \quad & \text{en } \Omega \\ u \quad \text{dada en } & \Omega^c \end{aligned} \tag{2.14}$$

que será combinado con pasos de difusión anisótropa que serán intercalado para dar estabilidad al algoritmo. Para el método de difusión anisótropa se utilizará el siguiente modelo de difusión

$$u_t = \varphi_\epsilon(x) \, |\nabla u| \, \nabla \cdot \frac{\nabla u}{|\nabla u|} \tag{2.15}$$

donde φ_ϵ es una función de corte que fuerza a que la ecuación de difusión funcione únicamente en Ω y $\nabla \cdot \left(\frac{\nabla u}{|\nabla u|} \right)$ es la curvatura a lo largo de los isófotos.
Esta ecuación de difusión ha sido muy utilizada para eliminar el ruido de una imagen mientras se conservan los bordes. En esta ocasión la ecuación compensa cualquier posibilidad de choque creada por la ecuación de transporte.

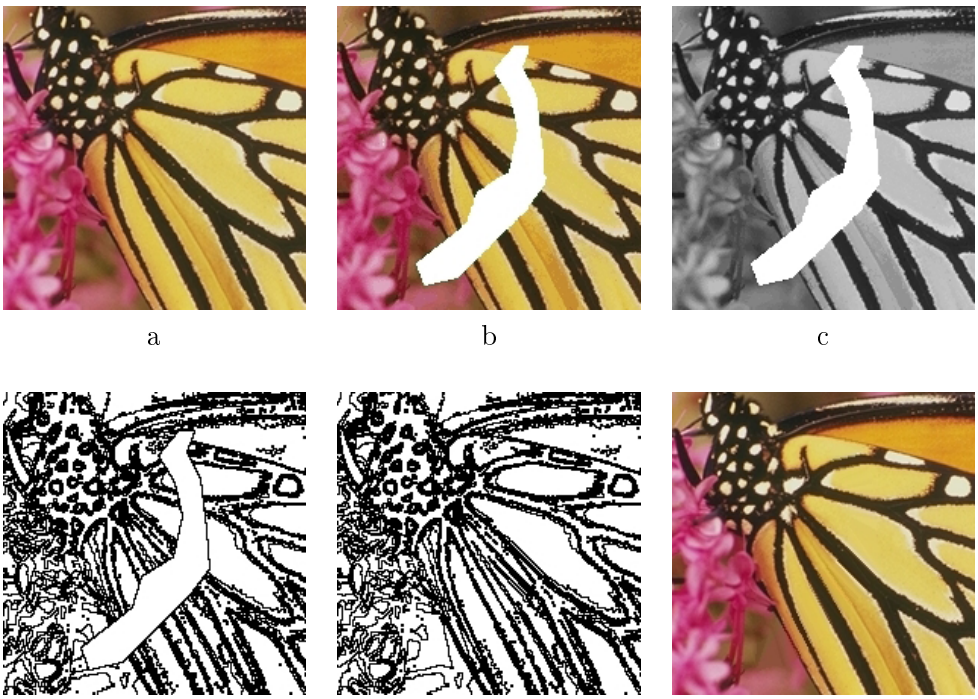


Figura 2.9: (a) Imagen original. (b) Imagen con una región oculta mediante el color blanco. (c) Canal de la luminancia con la región oculta. (d) Representación de algunos isófotos previos a la restauración. (e) Reconstrucción de los isófotos mediante el algoritmo de Masnou y Morel [23]. (f) Resultado final tras aplicar el mismo método a los canales de luminancia, tono y saturación.

El significado de la ecuación (2.14) viene dado en [6]. Δu es una medida de la suavidad de la imagen y los puntos estacionarios para la ecuación son aquellos para los cuales Δu es constante a lo largo de los isófotos inducidos por el campo de vectores $\nabla^{\perp} u$. La ecuación (2.14) no es una ecuación de transporte para Δu en su forma explícita, pero en su forma equivalente se puede reescribir como

$$u_t = -\nabla^{\perp}\Delta u \cdot \nabla u \tag{2.16}$$

lo cual sí que es una ecuación de transporte para u por la convección que general el campo $\nabla^{\perp}\Delta u$. Siguiendo el artículo de Bornemann y März [11], este campo se encuentra en las dirección de las curvas de nivel de Δu, las cuales están relacionadas con las fronteras de Marr-Hildreth. En efecto, los cruces por cero de Δu son la forma clásica de caracterizar bordes dada por el modelo de Marr y Hildreth. En otras palabras, como en el arte real, el método de Bertalmio y cía. [6] consiste en transportar las intensidades (los valores) de la imagen a lo largo de la dirección que indican los bordes, desde la frontera del dominio de *inpainting* Ω hacia el interior. La eficiencia de este enfoque se puede observar en la Figura 2.10.

Desde un punto de vista numérico, la ecuación de transporte y la ecuación de difusión anisótropa pueden ser implementadas mediante esquemas de diferencias finitas. Para las imágenes en color, el método puede ser aplicada de forma independiente a cada canal de

la imagen de cualquier representación clásica de luminancia o crominancia. En el método no hay restricción en la topología del dominio de *inpainting*.

Figura 2.10: Experimento tomado del artículo de Bertalmío y cía. [6]. Izquierda: Imagen original. Centro: Máscara definida por el usuario pintada en blanco. Derecha: Resultado del algoritmo de [6].

Existe otra perspectiva de este modelo propuesta por Bertalmio, Bertozzi y Sapiro en [7], donde se muestran las conexiones con las ecuaciones clásicas de Navier-Stokes para dinámica de fludos. De hecho, la ecuación que expresa el estado estacionario en [6]

$$\nabla^{\perp} u \cdot \nabla \Delta u = 0$$

es exactamente la ecuación que satisface el estado estacionario para flujos viscosos de fluídos incomprensibles del modelo de Navier-Stokes. Aunque la ecuación de difusión anisótropa (2.15) no es la contrapartida exacta para el término viscoso de difusión utilizado en el modelo de Navier-Stokes para flujos Newtonianos o incomprensibles. Todavía se puede aplicar y adaptar mucho conocimiento numérico en mecánica de fluidos para diseñar esquemas más estables y eficientes para *inpainting*. Los resultados en esta dirección se pueden observar en [7].

Oliverira y cía. en [9] propone un método rápido para realizar la restauración de una zona dañada debido a la lentitud con la que actuaba el método propuesto por Bertalmío en [6]. Esta restauración está enfocada a pequeñas regiones dañadas. La base del algoritmo reside en que las regiones dañadas pueden ser aproximadas mediante un proceso de difusión anisótropa que propague la información de $\partial\Omega$ en Ω.

En su versión más sencilla, el algoritmo consiste en iniciailzar Ω eliminando su información y convolucionando de forma repetida la región a restaurar con un *kernel* de difusión. Los kernels utilizados son similares a los de la Figura 2.11. El estado estacionario de este método se alcanzará cuando los píxeles pertenecientes al dominio Ω modifiquen sus valores por debajo de un umbral preestablecido. Aparte de esta condición de parada, también se propone especificar un número de iteraciones máximo previo a la ejecución del algoritmo

Realizar una convolución de una imagen con un *kernel gaussiano* es equivalente a realizar una difusión isótropa como indica la teoría de la ecuación lineal del calor. Este

a	b	a
b	0	b
a	b	a

c	c	c
c	0	c
c	c	c

Figura 2.11: *Kernels* propuestos por Oliveira y cía. en [9]. Dentro de estas tablas los valores representados corresponden con $a = 0{,}073253$, $b = 0{,}176765$, $c = 1/8 = 0{,}125$

algoritmo utiliza un *kernel* de media que sólo considera la contribución de los píxeles vecinos.

En [9] también se expone una variación del método para la preservación de los bordes. En esta variación, el usuario debe indicar una nueva máscara donde existan o debieran existir bordes las cuales nombra como "barreras de difusión". Los píxeles que integrarán esta máscara serán tratados como fronteras para el proceso de difusión.

Chan y Shen propusieron en [24] un modelo de primer orden para *inpainting* y reducción de ruido basado en la minimización conjunta de un término de fidelidad cuadrática fuera de Ω y un criterio de variación total en Ω, como puede ser la energía conjunta

$$\int_A |\nabla u| \ dx + \frac{\lambda}{2} \int_\Omega |u - u_0|^2 \ dx$$

con $A \supset\supset \Omega$ es el dominio de la imagen y λ es un multiplicador de Lagrange. La existencia de soluciones a este problema se consigue fácilmente mediante las propiedades de las funciones de variación acotada. Como para la implementación, Chan y Shen buscan puntos críticos para la energía utilizando el esquema iterativo de Gauss-Jacobi para el sistema lineal asociado a una aproximación de la ecuación de Euler-Lagrange por diferencias finitas.

Otros enfoques más recientes a la minimización de variación total con precisión de subpíxeles son los preferidos hoy en día. Desde el punto de vista fenomenológico, el modelo de Chan y Shen [24] rellena los candidatos con aquel que posea el menor de los isófotos posible. Es por eso que es más adecuado para dominios delgados o dispersos. Una resultado de este modelo está expuesto en la Figura 2.12.

Figura 2.12: Experimento extraído del artículo de Chan y Shen [24]. Izquierda: Imagen original. Derecha: Imagen tras la reducción de ruido y la eliminación del texto.

Si se vuelve a criterio expresado en la ecuación (2.13) de Masnou y Morel y se le añade una penalización similar en $\tilde{\Omega}$ tanto de la longitud como de la curvatura de todos los isófotos en la imagen u da lugar a dos formas equivalentes en el caso de que u sea lo

suficientemente suave (véase el artículo de Masnou y Morel [23]):

$$\int_{-\infty}^{+\infty} \int_{\{u=t\}\cap\tilde{\Omega}} (\alpha + \beta\,|\kappa|^p)\ ds\ dt = \int_{\tilde{Omega}} |\nabla u| \left(\alpha + \beta \left|\nabla \cdot \frac{\nabla u}{|\nabla u|}\right|^p\right)\ dx \qquad (2.17)$$

Hay muchos contribuyentes en el método de realizar una aproximación numérica para los puntos críticos de este criterio. Chan, Kang y Shen en [25] propusieron un método de cuarto orden dependiente temporalmente basado en la aproximación de la ecuación de Euler-Lagrange en el caso $p = 2$ utilizando diferencias finitas hacia atrás y una fórmula de módulo mínimo para estimar la curvatura. Este método evolutivo de tan alto orden sufre de una estabilidad bien conocida y de problemas de convergencia que son muy difíciles de tratar.

Un modelo ligeramente diferente de (2.17) es abordado por Ballester, Bertalmío, Caselles, Sapiro y Verdera en [26] utilizando un enfoque de relajación. La idea básica es reemplazar el término de segundo orden $\nabla \cdot \frac{\nabla u}{|\nabla u|}$ con un término de primer orden dependiente de una variable auxiliar.
Definiéndolo de forma más precisa, Ballester y cía. estudian en [26] la minimización de

$$\int_{\tilde{\Omega}} |\nabla \cdot \theta|^p\,(a + b\,|\nabla k * u|)\ dx + \alpha \int_{\tilde{\Omega}} (|\nabla u| - \theta \cdot \nabla u)\ dx$$

bajo la restricción de que θ es un campo vectorial con módulo por debajo de la unidad y preescrito con una componente normal en la frontera de $\tilde{\Omega}$ y u toma valores en el mismo rango que en Ω^c. Claramente, θ juega el factor de $\nabla u/\,|\nabla u|$ pero el nuevo criterio es mucho menos singular. k es un núcleo regularizador introducido por razones técnicas para asegurar la existencia de una pareja minimizadora (u, θ). La principal diferencia entre el criterio de relajación y (2.17) reside en la singularidad y en el término $\int_{\tilde{\Omega}} |\nabla \cdot \theta|^p\ \ dx$ que es más restrictivo que $\int_{\tilde{\Omega}} |\nabla u| \left|\nabla \cdot \frac{\nabla u}{|\nabla u|}\right|^p\ dx$ a pesar de la relajación.

Sin embargo, el nuevo modelo tiene una buena propiedad: se puede calcular bajo un métod de gradiente descendente con respecto a (u, θ) y da lugar a una pareja de ecuaciones de segundo orden cuya aproximación numérica es estándar. Un resultado obtenido con este modelo es el de la Figura 2.13.

El modelo Mumford-Shah-Euler de Esedoglu y Shen en [27] también es variacional. Combina el modelo de segmentación de imágenes de Mumford-Shah y el modelo de curvas elásticasd de Euler. Es decir, denotando u como una función continua a trozos débilmente suave, que posee un gradiente que es cuadrado integrable fuera del conjunto de discontinuidad $K \subset \tilde{\Omega}$, el criterio propuesto se establece como

$$\int_{\tilde{\Omega}\setminus K} |\nabla u|^2\ dx + \int_K \left(\alpha + \beta k^2\right)\ ds$$

En [27] Esedoglu y Shen exponen dos enfoques distintos para este criterio.
El primero, es un enfoque mediante *level sets* o curvas de nivel donde se representa K como la curva de nivel cero de una secuencia de funciones suaves que tienden a concentrarse y la derivación explícita mediante diferencias finitas de las ecuaciones de Euler-Lagrange asociadas con el criterio.
El segundo, es una aproximación de Γ-convergencia basada en el resultado original conjeturado por De Giorgi y probado por Schätzle. En ambos casos, el sistema final de ecuaciones

Figura 2.13: Izquierda: Imagen original. Derecha: Imagen restaurada utilizando el modelo propuesto por Ballester y cía. en [26].

discretas tiene cuarto orden, dando lugar otra vez a problemas relacionados con la estabilidad y la convergencia.

Más recientemente Esedoglu, Ruuth y Tsai en [28] abordaron el flujo numérico asociado con el modelo de Mumford-Shah-Euler utilizando un prometedor método basado en la umbralización y la convolución que es mucho más fácil de utilizar que las aproximaciones anteriores.

Tschumperlé propuso en [29] un modelo de difusión anisótropa de segundo orden para regularización de imágenes multivaluadas e *inpainting*. Dada una imagen u con valores en $\mathbb{R}^N$ conocidos fuera de Ω y comenzando por un *inpainting* rudo y grosero realizado mediante una copia de los valores de la frontera hacia el interior, los píxeles en el dominio de *inpainting* serán actualizados de forma iterativa de acurdo a una aproximación en diferencias finitas de las ecuaciones

$$\frac{\partial u_i}{\partial t} = \text{trace}(T\nabla^2 u_i),\ i \in \{1, \ldots, N\}$$

Donde T es un campo tensorial definido como

$$T = \frac{1}{(1+\lambda_{min}+\lambda_{max})^{\alpha_1}} v_{min} \otimes v_{min} + \frac{1}{(1+\lambda_{min}+\lambda_{max})^{\alpha_2}} v_{max} \otimes v_{max}$$

con $0 < \alpha_1 \ll \alpha_2$ y λ_{min}, λ_{max}, v_{min} y v_{max} son autovalores y autovectores de

$$G_\sigma * \sum_{i=1}^{N} \nabla u_i \otimes \nabla u_i$$

donde G_σ es un kernel gaussiano de suavizado y $\sum_{i=1}^{N} \nabla u_i \otimes \nabla u_i$ es un tensor de estructura clásico para representar de mejor forma la geometría local de u. La Figura 2.14 reproduce un experimento tomado del artículo de Tschumperlé [29].

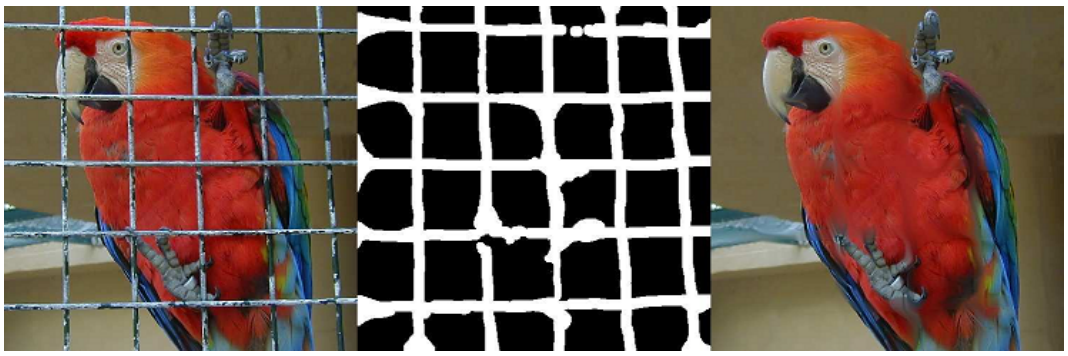


Figura 2.14: Ejemplo de inpainting utilizando el algoritmo de Tschumperlé [29]. Izquierda: Imagen original. Centro: Máscara de restauración definida por el usuario (color blanco). Derecha: Resultado del algoritmo.

La aproximación de Auroux y Masoudi en [30] utiliza las técnicas de EDPs que han sido desarrolladas para el problema del inverso de la conductividad en el contexto de detección de grietas. La conexión entre este tipo de detección y el *inpainting* es el siguiente: los bordes desaparecidos son modelados como grietas y se asume que la imagen es suave fuera de esos bordes. Dada una grieta se pueden obtener dos candidatos de la ecuación de Laplace con condiciones de frontera Neumann a lo largo de la grieta aplicando condiciones de tipo Dirichlet o Neumann a la frontera del dominio de *inpainting* $\partial\Omega$.
Las grietas óptimas serán aquellas para las cuales dos candidatos tienen las normas cuadráticas más similares y pueden ser encontrados a través de análisis topológico. Es decir, les corresponde un conjunto de puntos donde en la mayoría de ellos decrece la diferencia cuadrática. Tanto la localización de las gritas como las soluciones a trozos pueden ser encontradas utilizando esquemas en diferencias finitas rápidos y sencillos.

Telea [10] introdujo un algoritmo que rellenaba los "huecos" indicados por la región Ω en una única pasada por los píxeles de la imagen. El recorrido de los píxeles de la imagen lo realiza desde la frontera $\partial\Omega$ hacia el interior de ésta. Para establecer los valores utiliza la media de los valores conocidos cercanos al punto. El inconveniente de este método es que depende de la geometría de la región a restaurar Ω y no de la imagen, debido a este motivo los resultados no seguiran de forma correcta la orientación de los isófotos.

El algoritmo parte de que se tiene un conjunto de píxeles $\{x_1, \ldots, x_n\} \in \Omega$ ordenados para restaurarlos y define un vecindario de estos los píxeles como

$$B_\epsilon = \{y \in D \,/\, \|y - x\| \leq \epsilon\}$$

donde D representa el conjunto de los píxeles de la imagen que no se encuentran en el dominio de la máscara Ω.
Telea utiliza este vecindario B_ϵ para restaurar el valor de la imagen en el punto de la frontera $\partial\Omega$, pero no utiliza todos sino que define, a partir de B_ϵ, el vecindario de los píxeles que tienen un valor ya establecido. Este vecindario lo define como

$$B_\epsilon^{\angle}(x_k) = \{B_\epsilon(x_k) \,/\, \{x_k, \ldots, x_N\}\},\ k = \{1, \ldots, N\}$$

Entonces el algoritmo rellena los puntos de la imagen utilizando una media ponderada de los píxeles pertenecientes a $B_\epsilon^{\angle}$ de cada uno de los píxeles de la imagen inicial I^0 como

indica la siguiente fórmula

$$I(x_k) = \frac{\sum\limits_{y \in B_\epsilon^{\angle}(x_k)} w(x_k, y) I(y)}{\sum\limits_{y \in B_\epsilon^{\angle}(x_k)} w(x_k, y)},\ k = \{1, \ldots, N\}$$

donde $w(x, y) \geq 0$ son los pesos del algoritmo y se asume que

$$\sum_{y \in B_\epsilon^{\angle}(x)} w(x, y) > 0, x \in \Omega$$

Para el cálculo de los pesos $w(x, y)$ Telea realiza sugiere varios. Sin embargo, los resultados no varían demasiado. Si se toma el mas simple de ellos

$$w(x, y) = \frac{\|\nabla T(x) \cdot (x - y)\|^2}{\|x - y\|}$$

donde ∇ aproxima el gradiente.
La relación mediante la que se calcula este peso tiene el aporte en dos sentidos bien distintos: el numerador contribuye en mayor medida si $\vec{xy}$ se encuentra en la dirección de mayor descenso y el denominador aumentará el valor del peso $w(x, y)$ cuanto más cercanos estén ambos puntos.
Este algoritmo cumple una serie de propiedades deseables para la estabilidad ya que es lineal.

El relleno de los píxeles $x \in \Omega$ se realiza de acuerdo con la distancia euclídea $T(x)$ que los separa de la frontera $\partial\Omega$

$$T(x_i) < T(x_j) \Rightarrow i < j$$

Para determinar el valor de la prioridad de los píxeles se utiliza una lista ordenada por el valor de T calculado en el interior de Ω. El orden de la lista será realizado de forma incremental. El cómputo de este valor indicativo del orden se realiza mediante el método de marcha rápida (*Fast Marching Method*). Este método calcula T como una discretizacion de la distancia euclídea a la frontera $\partial\Omega$, $T(x) = \text{dist}(x, \partial\Omega)$. Esta función es la única solución del problema de tipo Dirichlet para la ecuación eikonal

$$\|\nabla T\| = 1 \,\text{en}\, \Omega,\ T|_{\partial\Omega} = 0$$

Finalmente, Bornemann y März propusieron en [11] un modelo de primer orden para transportar la información de la imagen a la región Ω a rellenar a lo largo de curvas integrales de un vector de coherencia que extiende en Ω las direcciones dominantes del gradiente de la imagen. Este campo de coherencia se define de forma explícita en cada punto cmo el autovector normalizado correspondiente con el mínimo autovalor de un tensor de estructura cuyo cálculo evita de forma cuidadosa los sesgos en el vecindario de la frontera de $\partial\Omega$.
Si se denota c como el campo de coherencia, Bornemann y März muestran que la ecuación $c \cdot \nabla u = 0$ con restricciones de frontera de tipo Dirichlet puede obtenerse como el límite de fuga de la viscosidad de un esquema de marcha rápida (utilizando el método de marcha rápida o *Fast Marching Method*, al igual que utilizó Telea [10]): los pixeles en Ω son sintetizados de uno en uno, de acuerdo a su distancia a la frontera $\partial\Omega$. El nuevo valor de un

píxel p es una combinación lineal de los valores ya conocidos y también de los generados en el momento de restaurar el píxel. La clave principal del método es la definición explícita de los pesos lineales de acuerdo con el campo de coherencia c.
A pesar de que el modelo de Bornemann y März requiere de un ajuste cuidadoso de cuatro parámetros, es mucho más rápido que el resto de enfoques en base a EDPs mencionados anteriormente y su resultados son muy buenos, como se muestra en la Figura 2.15.

Figura 2.15: Experimento de *inpainting* realizado por Bornemann y März en [11] con un tiempo de computación de 0.4 segundos. Izquierda: Imagen original. Centro: Máscara de restauración definida por el usuario en color blanco. Derecha: Resultado del algoritmo.

2.2.3.4. Combinación y extensión de los modelos basados en EDPs y los modelos basados en parches

En general, la mayoría de los métodos variacionales y los basados en EDPs que han sido nombrados previamente funcionan bien tanto en dominios finos como en dominios dispersos. Sin embargo, existe un inconveniente común en todos ellos: no son capaces de restaurar bien la textura y esto se hace muy visible en el caso de la restauración de dominios grandes. Esto se muestra en la Figura 2.14 donde se puede apreciar que el algoritmo no es capaz de recuperar la textura del loro.
Por otro lado, los métodos basados en parches no son capaces de manejar dominios de *inpainting* dispersos, como el que aparece en la Figura 2.16, donde no se puede encontrar ningún parche cuadrado que no este reducido a un punto.
Por el contrario, la mayoría de los métodos variacionales o basados en EDPs sí que son aplicables en esas situaciones, como en la Figura 2.16, donde se observa que el método propuesto por Masnou y Morel [23] da el resultado de la restauración.

Existen multitud de intentos de combinar de forma explícita los métodos basados en parches con los basados en EDPs para restaurar correctamente tanto la textura como las estructuras geométricas.
En [8], Bertalmío, Vese, Sapiro y Osher utilizan una descomposición aditiva de la imagen a ser restaurada. La descomposición se compone de dos componentes: una que contiene la información geométrica con toda la información de los bordes y otra componente que contiene la información sobre la textura. Entonces se utiliza el algoritmo de Efros y Leung expuesto en [13] para restaurar la textura de la imagen y para restaurar la geometría de la imagen se utiliza el método propuesto por Bertalmio y cía. en [6] (muchos trabajos posteriores también han propuesto otros métodos para la reconstrucción individual de cada una de las componentes).
La imagen final se obtendrá sumando las dos componentes, la de textura y la de geometría,

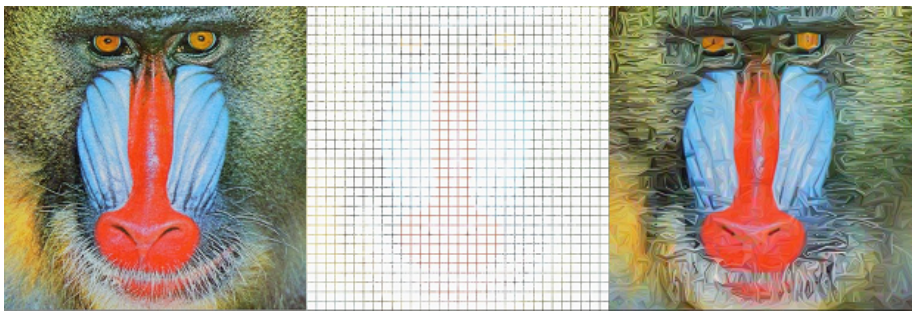


Figura 2.16: Izquierda: La imagen original del mandril. Centro: La imagen después de eliminar cuadrados de tamaño 15×15 píxeles (se han eliminado más del 87 % de los píxeles de la imagen). Derecha: La reconstrucción introducida por el método de Masnou y Morel [23] utilizando únicamente la información en las esquinas de los cuadrados.

ya restauradas. En las pocas situaciones donde la descomposición aditiva tiene sentido, este enfoque mejora el resultado y aumenta las aplicaciones del dominio de *inpainting*.

Komodakis y Tziritas en [31] combinan dos estrategias, la variacional y la basada en parches. Con ese objetivo definen una función de energía de inpanting sobre un grafo cuyos nodos son los centros de los parches sobre la imagen. Esta función de energía tiene dos términos, uno correspondiente a la síntesis de textura y otro correspondiente a la medida de similaridad de la superposición de dos parches vecinos (centrados en los nodos que son vecinos en el grafo). Mediante la minimización de esta función de energía con un algoritmo *Belief Propagation* en el que se asigna una etiqueta a cada nodo lo que equivaldría a copiar el parque correspondiente a la etiqueta en la posición del nodo.
Los resultados son muy buenos en una variedad de imágenes distintas (véase la Figura 2.2) y el método es rápido. Algunos problemas potenciales son la falta de seguridad de que el algoritmo iterativo converga a un mínimo global y que puedan aparecer artefactos visuales puesto que el método utiliza una malla fija y se copian los parches enteros para cada píxel de la máscara.

El trabajo de Drori, Cohen-Or y Yeshurun en [32] no involucra ninguna descomposición de geometría y textura, pero la búsqueda de vecindarios similares es guiada por una estimación aproximada de los valores restaurados utilizando un muestreo multiescala y una estrategia de convolución al igual que realiza Ogden y cía. en [22]. Además, a diferencia de muchos otros métodos basados en parches, el diccionario de parches válidos es enriquecido utilizando rotaciones, escalados y reflexiones. Si se desea ver un ejemplo, este está reflejado en la Figura 2.17.

2.2.3.5. Problemas abiertos

El problema de *inpainting* es un gran desafío muy complejo que todavía está lejos de poder llegar a solucionarse de una forma general y aceptable en cualquier circunstancia. Unos ejemplos de los problemas que se dan pueden observarse en la Figura 2.18. Uno de los grandes desafíos del problema es la forma de la región de *inpainting* Ω, la cual es arbitraria y es definida por el usuario.
Los métodos basados en parches son los que mejor funcionan como norma general, aunque

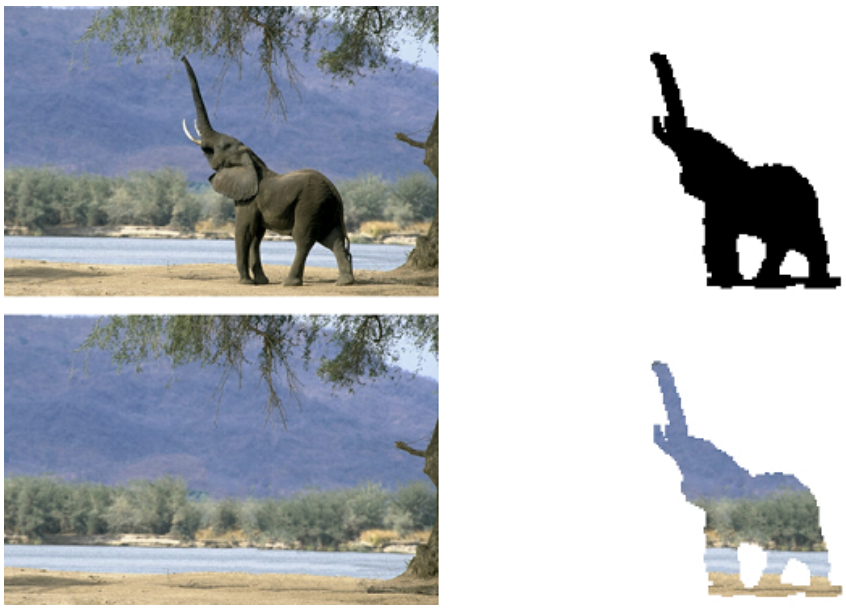
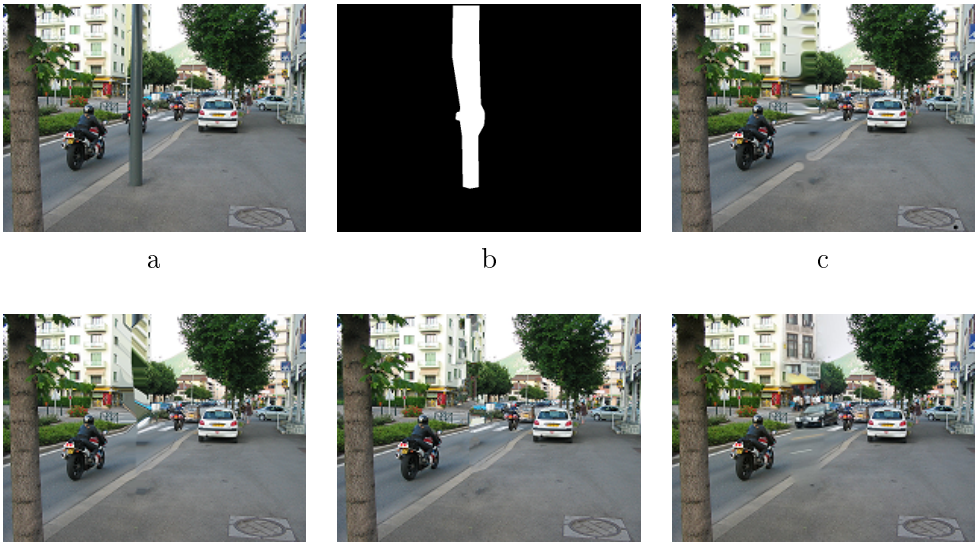


Figura 2.17: Experimento del artículo de Drori y cía. [32] mostrando su método basado en parches y en difusión multiescala. La imagen de arriba a la izquierda es la original. La de arriba a la derecha contiene la máscara definida por el usuario en color negro. Abajo a la izquierda se puede observar el resultado y a su derecha se muestra cómo han sido sintetizados los píxeles que ocupaban el lugar del elefante.

para algunas aplicaciones (como son regiones Ω muy dispersas o muy extensas) los métodos basados en la geometría son más adecuados.
Cuando el hueco en la imagen reside en una localización singular, donde la información que lo rodea no puede ser encontrada en ningún otro punto de la imagen, entonces los métodos basados en parches dan resultados muy pobres, a pesar de que se considere o no la geometría de la imagen.



a b c

d e f

Figura 2.18: Un ejemplo donde los métodos de *inpainting* no funcionan correctamente. (a) Imagen original extraída de la base de datos de Hays y Efros en [33]. (b) La máscara original con la que se indica la región a restaurar se trata de una modificación de la que utilizan Hays y Efros en su algoritmo. (c) Resultado del algoritmo de Tschumperlé [29]. (d) Resultado del algoritmo de Bornemann y März [11]. (e) Resultado del algoritmo de Criminisi [14]. (f) Resultado del algoritmo de Hays y Efros [33].

Capítulo 3

Métodos seleccionados

Como se ha podido observar en el capítulo anterior, existe una gran cantidad de métodos en el campo de *inpainting* partiendo de múltiples enfoques. Estos métodos aportan multitud de posibles soluciones al problema, pero algunos de ellos necesitan de un gran número de factores o parámetros para poder aplicarlos y otros requieren de una grandísima carga de trabajo para el procesador.

También se ha podido comprobar que en el campo del *filtrado de difusión anisótropa* existe una gran cantidad de métodos, pero éstos fueron creados poco a poco, conforme se introducían mejoras en el algoritmo ideado por Perona y Malik [1].

Lo que se va a tratar en este capítulo es la elección de los métodos elegidos y se realizará la explicación de los métodos elegidos. Estos métodos serán los que más adelante se implementarán en *Matlab* y finalmente introducirán mediante código *C* dentro de la aplicación móvil para *Android*.

3.1. Filtrado de difusión anisótropa

Los métodos dentro del campo del filtrado de difusión anisótropa han ido desarrollándose, mutando y mejorando desde la aparición del artículo de Perona y Malik [1], donde se establecieron las bases.

La primera de las implementaciones del modelo propueto por Perona y Malik fue realizada por Gerig en [37]. Como se puede observar los resultados son algo rudimentarios, pero también se debe tener en cuenta que la proposición del método ya fue en grandísimo paso dentro del campo del procesado de imágenes digitales.
Como ya se ha podido observar en el *estado del arte* de este conjunto de métodos, la técnica ha avanzado muchísimo con el paso de los años y en la actualidad se encuentra un gran número de técnicas adaptadas a los diferentes tipos de ruido existentes.

En esta sección se va a proceder a explicar las bases matemáticas del *filtrado de difusión anisótropa*. Inicialmente se va a realizar la explicación para comprender la base de la difusión anisótropa que propusieron Perona y Malik [1] para más tarde introducir el método semi-implícito desarrollado por Weickert en [2].

3.1.1. Modelo explícito

3.1.1.1. Base matemática

Se recuerda que Perona y Malik [1] enunciaron los criterios que se debían cumplir para obtener un buen modelo de descripción de imágenes multiescala. Estos criterios son:

1. Causalidad: Cuando disminuya la resolución no pueden generarse colores espúreos en la imagen.
2. Localización inmediata: En cada resolución los límites de las regiones deben ser distinguidos y deben coincidir con los originales.
3. Suavizado según zona: El suavizado debe ser mayor dentro de una región que con las regiones vecinas.

Teniéndolos en cuenta, se definió un nuevo modelo de filtrado de difusión que, lógicamente, tiene como base matemática la ecuación de difusión del calor. Esta ecuación expresada para el procesado de imágenes se representaría del siguiente modo:

$$\frac{\partial I(x,y,t)}{\partial t} = \text{div}\left(c(x,y,t)\nabla I(x,y,t)\right) \tag{3.1}$$

donde $I(x,y)$ representaría la función de la intensidad de la imagen, $I(x,y,t)$ es la evolución de la imagen en el tiempo y div es el *operador divergencia*, definido:

$$\text{div} f(\vec{x}) = \sum_{i=0}^{n-1} \frac{\partial f}{\partial x_i} \tag{3.2}$$

y el operador ∇ denota el *gradiente*:

$$\nabla f(\vec{x}) = \left(\frac{\partial f}{\partial x_0}, \ldots, \frac{\partial f}{\partial x_{n-1}}\right) \tag{3.3}$$

La función $c(x,y,t)$ se conoce con el nombre de *coeficiente de difusión* y junto con el *gradiente* describe el flujo:

$$\Phi = c(x,y,t)\nabla I(x,y,t) \tag{3.4}$$

Koenderink estableció que el coeficiente c debía de ser una constante independiente de la localización espacial. Es este el caso en el que se realiza un filtrado lineal, dando lugar a un flujo isótropo y es cuando además de eliminar ruido, se eliminan detalles.
Perona y Malik establecieron que el coeficiente c debía variar según su localización espacial. Así se obtiene entonces un flujo anisótropo correspondiente a un filtrado no lineal.

3.1.1.2. Coeficiente de difusión

La elección del *coeficiente de difusión* debe ser tal que se cumplan los nuevos criterios establecidos, y que no se deje de cumplir el criterio de causalidad. Como se ha dicho, debe ser preferente el suavizado dentro de una región al suavizado entre regiones. Esto podría llevar a pensar que el *coeficiente de difusión* debe ser 1 en el interior de cada región y 0 en

las fronteras. De esta manera se harían más diferenciables los bordes y delimitaciones. Por supuesto, no se desconoce el límite de cada región a priori, en ese caso el problema ya estaría resuelto. Lo que se puede hacer es estimar una localización apropiada de los bordes para la escala correspondiente.

Si definimos $E(x, y, t)$ como un estimador, que idealmente debe cumplir las siguientes propiedades:

1. $E(x, y, t) = 0$ en el interior de cada región
2. $E(x, y, t) = K\Delta e(x, y, t)$ en cada punto correspondiente a un borde de la imagen, donde e es un vector unitario normal al borde y K es el *contraste local* del borde, la diferencia existente entre las intensidades de la imagen existentes a la derecha e izquierda del punto.

Si se puede definir un estimador para la imagen, entonces se puede escoger el *coeficiente de difusión* $c(x, y, t)$ como una función dependiente de la magnitud del estimador E.

$$c = g(\|E\|)$$

$g(\cdot)$ debe de ser una función no negativa y monótona decreciente con $g(0) = 1$. Así el proceso de difusión se producirá principalmente en el interior de las regiones y no afectará a las fronteras, donde la magnitud de E es grande.

Es intuitivo que el proceso de difusión satisface los tres criterios establecidos previamente. El grado en que se cumplen viene determinado por la exactitud con la que el estimador E defina la posición de los bordes de la imagen. Definir exactamente la posición de los bordes requiere mucha carga computacional y algoritmos muy complicados, por tanto es necesario buscar alguna manera sencilla que permita estimar aproximadamente dichos bordes. Afortunadamente se obtienen muy buenos resultados si simplemente se define el estimador como el *gradiente* del brillo de la imagen.

$$E(x, y, t) = \nabla I(x, y, t)$$

En resumen, hay muchas posibilidades para la elección de la función g, la más obvia es la función bivaluada como antes se ha comentado, y de todas ellas una opción bastante buena es la de definir g como una función del *gradiente* de la intensidad de la imagen. Y esta función debe ser monótona decreciente.

3.1.1.3. Propiedades de la difusión anisótropa

Se ha visto que el primer criterio que debe cumplir la difusión anisótropa es el criterio de causalidad. Este criterio establece que no se deben introducir nuevos detalles en la imagen al pasar de una resolución fina a una menos fina. También se puede definir diciendo que todos los máximos y todos los mínimos de la función intensidad de la imagen existentes en las distintas resoluciones deben pertenecer a la imagen original.

La ecuación de difusión (3.1) es un caso particular de ecuación elíptica. Estas ecuaciones satisfacen el *principio del máximo*, que establece que todos los máximos de la solución de la ecuación en el espacio y en el tiempo pertenecen a la condición inicial (imagen original). $I(x, y, t)$ obedece este principio debido a que el *coeficiente de difusión* nunca es negativo y

a que es diferenciable. Además, si las condiciones de frontera son adiabáticas (por lo que se establece que no puede haber ningún intercambio térmico a través de ella) los máximos sólo pertenecen a la condición inicial.
También se puede demostrar sencillamente que se cumple el *principio del mínimo*. Por lo tanto, se cumple el criterio de causalidad.

Con las técnicas de filtrado convencionales como el filtrado paso bajo y el filtrado basado en la difusión lineal el precio que se paga por la eliminación de ruido y mejora de resolución es la difuminación de los bordes. Esto hace que su detección y localización sea más difícil. Para la reconstrucción y mejora de los bordes de la imagen difuminada se puede hacer un filtrado paso alto o filtrar mediante el proceso de difusión hacia atrás en el tiempo. Esto tiene el problema de que en la mayoría de los casos se llega a un estado inestable.

En el caso de la difusión anisótropa si se escoge de forma apropiada el *coeficiente de difusión* se puede hacer que la difusión anisótropa mejore los bordes de la imagen, filtrando hacia adelante en el tiempo, con la estabilidad que garantiza el *principio del máximo*.

Si se modela un borde como una función escalón convolucionada con una gaussiana y se asume que el borde está alineado con el eje de coordenadas y, la expresión del *operador divergencia* se puede simplificar a:

$$\operatorname{div}(c(x,y,t)\nabla I) = \frac{\partial(c(x,y,t)I_x)}{\partial x}$$

Si según lo aconsejado se escoge c como una función del *gradiente* de la intensidad de la imagen I:

$$c(x,y,t) = g(I_x(x,y,t))$$

Y si denotamos como ya se ha visto antes el flujo $\Phi = cI_x$, entonces la ecuación de difusión (3.1) para una dimensión se escribiría del siguiente modo:

$$I_t = \frac{\partial \Phi(I_x)}{\partial x} = \Phi'(I_x)I_{xx} \tag{3.5}$$

Lo que interesa es la variación en el tiempo de la pendiente del borde: $\frac{\partial I_x}{\partial t}$. Si el coeficiente de difusión es mayor que cero ($c > 0$) la función $I(\cdot)$ se suaviza, y el orden de la diferenciación debe ser invertido:

$$\frac{\partial I_x}{\partial t} = \frac{\partial I_t}{\partial x} = \frac{\partial}{\partial t}\left(\frac{\partial \Phi(I_x)}{\partial x}\right) = \Phi'' I_{xx}^2 + \Phi' I_{xxx}$$

Suponiendo que el borde está orientado de modo que $I_x > 0$. En el punto de inflexión $I_{xx} = 0$, e $I_{xxx} \ll 0$ desde el punto de inflexión correspondiente al punto de máxima pendiente. Entonces en las vecindades del punto de inflexión $\frac{\partial(I_x)}{\partial t}$ tiene el signo contrario a $\Phi(I_x)$. Si $\Phi'(I_x) > 0$ la pendiente del borde decrecerá con el transcurso del tiempo y si, por el contrario, $\Phi'(I_x) < 0$ la pendiente se incrementará con el transcurso del tiempo. Es decir, es en este último caso cuando los bordes se harán más diferenciables.

Mediante el desarrollo que se acaba de hacer, se demuestra lo que anteriormente se había dicho: la función g debe ser monótona decreciente. Así se consigue que los bordes sean más afilados y por tanto se facilite la realización de operaciones posteriores. Hay que aclarar que aunque en la vecindad a la región de más pendiente de un borde quizás se produzca el

proceso de difusión hacia atrás en el tiempo debido a que $\Phi'(I_x)$ es negativa (3.5), y como se sabe esto podría llevar a estados inestables y a la amplificación de ruido, este hecho no es preocupante, ya que el *principio del máximo* garantiza que estos fenómenos no se produzcan. Experimentalmente se ha observado que en esas áreas en las que $\Phi'(I_x) < 0$ el proceso permanece estable.

Perona y Malik en [1] propusieron dos funciones para el coeficiente de difusión:

$$c_1(\vec{x},t) = \exp\left(-\left(\frac{|\nabla I(\vec{x},t)|}{K}\right)^2\right) \tag{3.6}$$

$$c_2(\vec{x},t) = \frac{1}{1+\left(\frac{|\nabla I(\vec{x},t)|}{K}\right)^2} \tag{3.7}$$

El *coeficiente de difusión* decrece monótonamente a medida que el *gradiente* aumenta. Cuando el *gradiente* es grande la difusión, por tanto, que se va a producir va a ser pequeña, y viceversa.

El comportamiento de estas dos funciones es diferente, la primera (3.6) va dar preferencia a los bordes con contraste alto frente a los de contraste bajo y la segunda (3.7) va dar preferencia a las regiones anchas frente a las estrechas. En cuanto al parámetro K, éste va influir de manera importante en el grado de difusión. A medida que éste es mayor la difusión es mayor para el mismo *gradiente*. Su valor va a depender de cual sea la aplicación concreta, y éste será elegido según el nivel de ruido y la intensidad de los bordes existentes. Este valor, para la realización del proceso de difusión, puede fijarse de antemano, así un valor adecuado, por ejemplo, es uno comprendido en el rango $1{,}5\ \sigma_{noise} < K < 2{,}0\ \sigma_{noise}$, o puede calcularse de alguna manera en cada iteración y por tanto ir variando y ajustándose al estado de la imagen en cada momento.

El máximo flujo se genera cuando ∇I es igual a K. Cuando decrece por debajo de K el flujo tiende a cero, ya que en las regiones homogéneas existe un flujo mínimo o nulo. Por encima de K el flujo otra vez decrece a cero, haciendo así que la difusión se pare en aquellas zonas con grandes gradientes.

3.1.2. Modelo semi-implícito

La aparición del modelo semi-implícito presenta un gran avance en el campo del procesado de imagen. Este modelo está basado en los mismos principios que el filtro propuesto por Perona y Malik [1] y sus resultados son muchísimo mejores. Los dos grandes avances presentados por el método propuesto por Weickert [2] con respecto al modelo anterior fue la introducción de un Tensor de Difusión para dirimir la cantidad de difusión que hay que realizar en cada una de las direcciones y la realización de un esquema semi-implícito de difusión incondicionalmente estable.

Estos dos grandes avances presentandos, los buenos resultados y la no excesiva complejidad del método fueron las razones por las cuales se eligió este método para realizar su implementación. Este método será expuesto a continuación.

3.1.2.1. Región de difusión

Para este filtrado se parte de un dominio rectangular $\Omega := (0, a_1) \times (0, a_2)$ con frontera $\Gamma := \partial\Omega$ y la imagen esté representada por $f \in L^\infty(\Omega)$ donde se da el problema de valores iniciales

$$\begin{aligned} \partial_t u = \operatorname{div}(D\nabla u) \quad &\text{en} \quad \Omega \times (0, \infty) \\ u(x, 0) = f(x) \quad &\text{en} \quad \Omega \\ \langle D\nabla u, n\rangle = 0 \quad &\text{en} \quad \Gamma \times (0, \infty) \end{aligned}$$

donde, n denota la normal saliente y $\langle ., . \rangle$ es el producto escalar (también denominado producto interno) en $\mathbb{R}^2$.

3.1.2.2. Tensor de Estructura

Para adaptar el tensor de difusión $D \in \mathbb{R}^{2\times 2}$ a la estructura local de la imagen, generalmente se utiliza un estimador ∇u_σ. Se definirá u_σ como

$$u_\sigma(x, t) := (K_\sigma * \tilde{u}(., t))(x)$$

donde $\sigma > 0$ y $\tilde{u}$ denota una extension de u de Ω a $\mathbb{R}^2$, replicando los valores de la frontera hacia el exterior de Ω.

Con todo ello se debe utilizar un descriptor más general que utilice la información proporcionada por el detector de bordes ∇u_σ y que permita extraer una mejor información sobre la estructura de la imagen.

Dicho descriptor servirá para identificar características de la imagen tales como son esquinas o para medir la coherencia local de las estructuras se define el tensor de estructura.

El tensor de estructura se utilizará para estudiar orientaciones. Se deben identificar los gradientes que difieren sólo por su signo, que aunque tienen la misma dirección sus sentidos son opuestos. Para ello, se calculará la matriz J_0

$$J_0(\nabla u) := \nabla u_\sigma \otimes \nabla u_\sigma := \nabla u_\sigma \nabla {u_\sigma}^T$$

La matriz definida por el tensor de estructura tiene una base ortonormal de autovectores v_1 y v_2 tal que $v_1 \parallel \nabla u_\sigma$ y $v_2 \perp \nabla u_\sigma$. Los autovalores correspondientes, $|\nabla u_\sigma|^2$ y 0, proporcionan el contraste en las direcciones indicadas por los autovectores.

Realizando una media de la información proporcionada por J_0, mediante una convolución por componentes con una gaussiana K_ρ de desviación típica $\rho > 0$, se obtiene el tensor de estructura

$$J_\rho(\nabla u) := K_\rho * (\nabla u_\sigma \otimes \nabla u_\sigma) := K_\rho * \left(\nabla u_\sigma \nabla {u_\sigma}^T\right) \tag{3.8}$$

Autovalores y autovectores

La matriz

$$J_\rho = \begin{pmatrix} j_{11} & j_{12} \\ j_{12} & j_{22} \end{pmatrix}$$

es simétrica definida positiva y posee vectores ortonormales v_1 y v_2 con

$$v_1 \parallel \begin{pmatrix} 2j_{12} \\ j_{22} - j_{11} + \sqrt{(j_{11} - j_{22})^2 + 4j_{12}^2} \end{pmatrix}$$

y, como $v_1 \perp v_2$, entonces se tendrá

$$v_2 \parallel \begin{pmatrix} j_{11} - j_{22} - \sqrt{(j_{11} - j_{22})^2 + 4j_{12}^2} \\ 2j_{12} \end{pmatrix}$$

Los autovalores correspondientes μ_1 y μ_2 son dados por

$$\mu_1 = \frac{1}{2}\left(j_{11} + j_{22} + \sqrt{(j_{11} - j_{22})^2 + 4j_{12}^2}\right)$$
$$\mu_2 = \frac{1}{2}\left(j_{11} + j_{22} - \sqrt{(j_{11} - j_{22})^2 + 4j_{12}^2}\right)$$

Éstos autovalores indican la media del contraste del campo en las direcciones de los autovectores dentro de un vecindario de tamaño $O(\rho)$.

Realizar el suavizado inicial ∇u_σ hace que el tensor de estructura sea insensible al ruido. También pasará por alto los detalles irrelevantes, los cuales serán eliminados con este suavizado. Idénticamente se harán irrelevantes detalles que sean de menor tamaño que la escala de integración $O(\sigma)$.

Debido a que $\mu_1 \geq \mu_2 \geq 0$, se puede observar que la orientación de v_1 es la que indica el mayor gradiente y v_2 da la orientación local preferida, la dirección más coherente para realizar la difusión.

Los autovalores μ_1 y μ_2 pueden ser utilizados como descriptores de la estructura local. Las diferentes áreas estarán caracterizadas por

- $\mu_1 = \mu_2 = 0 \rightarrow$ Serán áreas planas, que no contengan detalles.
- $\mu_1 \gg \mu_2 = 0 \rightarrow$ Serán las áreas que tengan bordes muy marcados.
- $\mu_1 \geq \mu_2 \gg 0 \rightarrow$ Serán esquinas o zonas que posean gran curvatura.

3.1.2.3. Problema continuo

Con todo lo expuesto hasta el momento, se puede introducir ahora el problema P_c reescribiendo el problema inicial de la siguiente manera:

Asumiendo que $f \in L^\infty(\Omega)$, $\rho >= 0$, $\theta > 0$ y $T > 0$, se puede reconsiderar el problema como

$$\begin{aligned} \partial_t u = \operatorname{div}(D(J_\rho(\nabla u_\sigma))\nabla u) \quad &\text{en} \quad \Omega \times (0, \infty) \\ u(x, 0) = f(x) \quad &\text{en} \quad \Omega \\ \langle D(J_\rho(\nabla u_\sigma))\nabla u, n\rangle = 0 \quad &\text{en} \quad \Gamma \times (0, \infty) \end{aligned}$$

donde el tensor de difusión $D = (d_{ij})$ debe satisfacer las siguientes propiedades:

1. Suavidad
   $D \in C^\infty(\mathbb{R}^{2\times 2}, \mathbb{R}^{2\times 2})$

2. Simetría
   $d_1 2(J) = d_2 1(J)$ para todas las matrices simétricas $J \in \mathbb{R}^{2\times 2}$.

3. Definida positiva y uniforme
   Para todo $w \in L^\infty(\Omega, \mathbb{R}^2)$ con $|w(x)| \leq K$ en $\bar{\Omega}$, existe un límite inferior positivo $\nu(k)$ para los autovalores de $D(J_\rho(w))$

Este problema cumple varias premisas:

- Es un problema "bien puesto" y tiene solución única.
- La solución es regular.
- Los valores de la solución se mantienen acotados.

3.1.2.4. Tensor de difusión

Ahora, a partir de los autovalores μ_1 y μ_2 y los autovectores correspondientes v_1 y v_2 calculados anteriormente a partir del tensor de estructura se puede calcular el tensor de difusión.

Puesto que el tensor de difusión debe reflejar la estructura local de la imagen, éste debe ser elegido de tal forma que el conjunto esté formado por los autovectores v_1 y v_2. La elección de los autovalores correspondientes λ_1, λ_2 depende del objetivo que quiera tener el filtro.

Como ejemplo de los objetivos del filtro en [2] se puede observar que utiliza este filtro con dos objetivos bien distintos. El primero de ellos es eliminar ruido de las imágenes y conseguir obtener una imagen con menos tonos cuyo objetivo puede ser una sementación final mucho más sencilla. El segundo de los objetivos es realizar realce de grandes detalles en la imagen.

Si se desea, como es el caso, suavizar el interior de regiones de la misma tonalidad y no permitir la difusion a través de bordes. Con este objetivo, siguiendo [3], se puede reducir la difusividad perpendicular a los bordes, la cual es dada por μ_1. Para solventar este apartado, se realizará la siguiente elección: $m \in \mathbb{N}$, $C_m > 0$ y $\lambda > 0$.

$$\begin{aligned} \lambda_1(\mu_1) &:= g(\mu_1) \\ \lambda_2 &:= 1 \end{aligned}$$

con

$$g(s) := \begin{cases} 1 & (s \leq 0) \\ 1 - \exp\left(\frac{-C_m}{(s/\lambda)^m}\right) & (s > 0) \end{cases}$$

Esta función exponencialmente decreciente es elegida para que se mantenga la suavidad. Al estar limitado ∇u_σ en $\Omega \times [0, \infty)$ y $\mu_1 = |\nabla u_\sigma|^2$, se sabe que el carácter de D va a ser definido positivo automáticamente.

La constante C_m es calculada de forma que el flujo $\Phi(s) = sg(s)$ es creciente en el intervalo $[0, \lambda]$ y decreciente en (λ, ∞). Esto se puede observar como una regularización anisótropa del modelo Perona-Malik.

La elección de los parámetros es idéntica a la que se realiza en [2] y muy similar a la presentada en [3].
Estos parámetros serán $m := 4$ y $C_4 = 3{,}31488$. Esta elección aporta buenos resultados en el plano visual y es la que se utilizará en el proceso. Puesto que en esta seccion sólo se esta interesado en la difusión que mantiene los bordes y que elimine el ruido, se puede establecer la escala de integración del tensor de estructura $\rho = 0$.

Así el esquema utilizado para conseguir eliminar el ruido seguirá mayoritariamente los vectores de μ_2, que son perpendiculares a las direcciones de máximo gradiente de la función. Y la matriz D se obtendrá como

$$\begin{aligned} D &= \begin{pmatrix} v_{1_x} & v_{2_x} \\ v_{1_y} & v_{2_y} \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} v_{1_x} & v_{2_x} \\ v_{1_y} & v_{2_y} \end{pmatrix}^T = \\ &= \begin{pmatrix} \lambda_1 {v_{1_x}}^2 + \lambda_2 {v_{2_x}}^2 & \lambda_1 v_{1_x} v_{1_y} + \lambda_2 v_{2_x} v_{2_y} \\ \lambda_1 v_{1_x} v_{1_y} + \lambda_2 v_{2_x} v_{2_y} & \lambda_1 {v_{1_y}}^2 + \lambda_2 {v_{2_y}}^2 \end{pmatrix} \end{aligned}$$

Y estos serán los coeficientes que más adelante se aplicarán en el esquema numérico. Renombraremos los elementos del tensor de difusión como:

$$D = \begin{pmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{pmatrix}$$

de donde se extrae que

$$\begin{aligned} D_{xx} &= \lambda_1 {v_{1_x}}^2 + \lambda_2 {v_{2_x}}^2 \\ D_{xy} &= \lambda_1 v_{1_x} v_{1_y} + \lambda_2 v_{2_x} v_{2_y} \\ D_{yy} &= \lambda_1 {v_{1_y}}^2 + \lambda_2 {v_{2_y}}^2 \end{aligned}$$

Por último, la resolución del sistema se realizará de forma numérica, lo cual será tratado más adelante.

3.2. *Inpainting*

Al ser un número tan grande de métodos se han estudiado varios métodos que correspondían con la filosofía de la programación sobre dispotivos móviles. Esta filosofía posee una máxima: el algoritmo debe ser ágil y consumir el mínimo número de recursos posible debido a la poca potencia que atesoran los dispositivos móviles en comparación con los equipos de procesamiento u ordenadores que se utilizan hoy en día.

Por estos motivos se descartaron rápidamente métodos en los que se realiza un gran uso del procesador y que son extremadamente lentos. También se descartaron los métodos de tipo *sparse* debido a la gran complejidad que entrañan, quedando en la práctica únicamente los métodos variacionales y los que hacen uso de EDPs. Tras leer varios de los artículos de este grupo se eligió el método de Bertalmío [6] debido a su sencillez y a que los tiempos de procesamiento necesarios en su tiempo eran relativamente cortos, con lo cual en la

actualidad lo serán todavía más. El inconveniente principal de este método es que no realiza la síntesis de texturas, por lo cual los resultados pueden ser pobres en los casos donde las regiones Ω en las que realizar el *inpainting* sean muy grandes.

3.2.1. Contexto del método

Cualquier método de *inpainting* es un proceso que trata de restaurar pinturas o imágenes dañadas y eliminar o reemplazar objectos seleccionados en imágenes sin dejar rastro del proceso seguido. Esta técnica es muy utilizada para devolver la unidad y la coherencia de una imagen o para hacer más legibles y verosímiles ciertas partes de la imagen.

El algoritmo que se utiliza generalmente rellena las regiones indicadas por el usuario mediante la información que las rodea. El algoritmo de refinado que se va a utilizar prolonga las líneas curvándolas hacia el interior de la región, otros simplemente las prolongan. De la misma forma también va rellenando poco a poco las regiones más planas de la imagen. La forma de trabajar del algoritmo de refinado en la recomposición de la imagen es la típica forma de restauración de las obras de arte.

Las aplicaciones más extendidas de este método son la restauración de viejas fotografías, la eliminación de datos sobreimpresos en las imágenes y la eliminación de objetos completos de la imagen como pueden ser micrófonos o cables en efectos especiales.

En el proceso de restauración se deberán tener en cuenta ciertos datos iniciales. Se partirá incialmente de una región Ω, la cual va a ser restaurada, y se nombrará como $\partial\Omega$ a su frontera.

Para situarnos en el contexto del proceso, se parte de la base de que el proceso de reconstrucción es completamente automático. El usuario lo único que debe proveer al algoritmo es la imagen donde se quiere realizar la reconstrucción y una máscara que indique la region Ω.

Acontinuación se va a exponer el método utilizado para la reconstrucción de imágenes. Este método sufre dos procesos bien distintos. Inicialmente se realizará una aproximación grosera a la solución que se realizará mediante un proceso muy rápido. Después, poco a poco se refinará con un método más elaborado y con mejores resultados. Con el objeto de realizar el primer cometido se utilizará una modificación del método propuesto por Telea [10] y con el del segundo se utilizará el algoritmo propuesto por Bertalmío [6].

Por otro lado, se debe explicar que la elección de la combinación de estos dos métodos es debida a que presentan unos resultados similares al método de Bertalmío [6], que son muy buenos, diminuyendo en gran medida los tiempos de procesado necesarios y en algunos casos, mejorando los resultados.

3.2.2. Método

En todos los casos se parte de una región Ω, la cual se va a repintar, y $\partial\Omega$ será su frontera. Esta región Ω se encontrará confinada dentro de la imagen inicial I_0, tal que

$$I_0(i,j) : [0,M] \times [0,N] \rightarrow \mathbb{R}$$

y

$$[0, M] \times [0, N] \subset \mathbb{N} \times \mathbb{N}$$

donde se supone que la imagen tiene un formato inicial de escala de grises. Más adelante, dentro de los detalles de la implementación del método, se tratará el procesado de imágenes a color.

Como el método propuesto por [6] era un poco lento para poder implementarlo en dispositivos móviles, se propone realizar inicialmente una aproximación grosera a la solución para refinarla a continuación utilizando el método propuesto por Bertalmío.

3.2.2.1. Aproximación de la solución

Utilizando parte de las ideas propuestas por Telea [10] se va a exponer un método rápido para realizar una aproximación grosera a la solución.

El método propuesto introducirá los valores que rodean a la región a restaurar Ω en su interior pasando únicamente una vez por cada píxel a restaurar, como se realiza en [10]. Aunque existirá una diferencia sustancial con éste ya que el orden de rellenado de los píxeles no estará indicado por la ecuación *eikonal*, sino que se implementará mediante una estrategia *onion-peel* o piel de cebolla.

La estrategia *onion-peel* toma como región inicial a restaurar $\Omega^0 := \Omega$. Se extraerán los píxeles pertenecientes a la frontera de la región $\partial\Omega^0$ y se resturarán. En el siguiente paso Ω^0 se contraerá, eleminando de su región los píxeles que ya hayan sido restaurados y dará lugar a Ω^1.
Este proceso de extracción, restauración y contracción de los píxeles de la frontera de Ω^i se realizará de forma iterativa, donde Ω^i es la máscara en la iteración i-ésima. De esta forma se irán restaurando los píxeles por capas, desde la frontera $\partial\Omega^0$ hacia el interior de ésta. La parada de este proceso iterativo se realizará cuando se llegue a un estado en el que Ω^i ya no contenga píxeles.

Para restaurar cada uno de los píxeles se utilizará un proceso similar al realizado por Telea en [10]. Para ello se utilizará una media ponderada de los píxeles pertenecientes al vecindario del punto utilizando la fórmula

$$I(x_k) = \frac{\sum_{y \in B_\epsilon(x_k)} w(x_k, y) I(y)}{\sum_{y \in B_\epsilon(x_k)} w(x_k, y)}, \; k = \{1, \ldots, N\} \tag{3.9}$$

donde $B_\epsilon(x_k)$ indica el vecindario de tamaño ϵ alrededor del punto x_k, w es la función de influencia (o peso) que tendrá el píxel y para restaurar el píxel x_k y k es el índice mediante el que se recorrerán todos los píxeles que pertenezcan a la frontera $\partial\Omega^i$.
La función de influencia estará definida de tal forma que tenga dos cometidos. El primero de esos cometidos será evaluar si el píxel pertenece o no a Ω^i. En el caso de que pertenezca al área que todavía se tiene que restaurar, la influencia del píxel será nula. En el caso contrario, el píxel sí que tendrá una influencia en el píxel a restaurar. El segundo cometido será asignar un peso dependiente de la distancia a la que se encuentre el píxel de la frontera, cuanto más cercano sea el píxel, más influencia tendrá en el píxel a restaurar.

El principal inconveniente de este método es el resultado. Este resultado no continúa la dirección de entrada de los isófotos en $\partial\Omega$, en vez de eso, prolonga los isófotos en las

direcciones normales a $\partial\Omega^i$, en este punto cabe recordar que la región Ω introducida por el usuario es arbitraria, por lo tanto, podrán producirse artefactos y formas extrañas en la imagen.
Otro inconveniente es la falta de conectividad en los bordes que se encuentra en los puntos centrales de la región Ω.
En contrapartida a estos inconvenientes, el gran beneficio de este método es su gran velocidad, puesto que el proceso es muy rápido debido a que el procesado realiza un recorrido que pasa una única vez por cada píxel.

3.2.2.2. Refinado de la solución

La técnica propuesta por Bertalmío en [6] prolonga las líneas isófotas (las que tienen el mismo nivel de tonalidad o nivel de gris) que llegan a $\partial\Omega$ manteniendo el ángulo de llegada. Además, también ayuda a rectificar los ángulos de entrada en Ω introducidos por la aproximación inicial.

El "repintado" se realiza desde la frontera hacia el interior de Ω. Mientras tanto se irán curvando las líneas de prolongación de forma progresiva para prevenir que se crucen unas con otras.

El proceso de evolución del algoritmo es iterativo. Durante esta iteración se creará una secuencia de imágenes que hará evolucionar I_0 hasta I_R, el cual será el resultado de la evolución. De esta forma el proceso se definirá como

$$I(i,j,n) : [0,M] \times [0,N] \times \mathbb{N} \to \mathbb{R}$$

tal que

$$I(i,j,0) = I_0(i,j)$$

y

$$I_R(i,j) = \lim_{n\to\infty} I(i,j,n)$$

Con todo ello, la expresión general del algoritmo será la siguiente:

$$I^{n+1}(i,j) = I^n(i,j) + \Delta t\, I_t^n(i,j), \forall (i,j) \in \Omega$$

donde

- Los superíndices n indican el tiempo de ejecución del algoritmo o el número de iteración que se está ejecutando.
- (i,j) son las coordenadas de la imagen.
- Δt es el ratio de mejora o la velocidad de variación.
- $I_t^n(i,j)$ es la actualización o variación de $I^n(i,j)$.

La evolución de esta imagen sólo se ejecuta dentro de la región Ω. El resto de los píxeles de la imagen permanecerán inalterados durante todo el proceso.

En este esquema la imagen $I^{n+1}(i,j)$ es una versión "mejorada" de $I^n(i,j)$. Dicha "mejora" viene dada por $I_t^n(i,j)$.

Para diseñar $I_t^n(i,j)$ es necesario continuar las líneas de llegada a la frontera $\partial\Omega$ de la región Ω, la cual se va a repintar. En otras palabras, se necesita propagar información del exterior de Ω hacia su interior.

Si se denomina $L^n(i,j)$ como la información que se desea propagar y $\overrightarrow{N^n}$ la dirección de propagación, entonces se tendrá que

$$I_t^n(i,j) = \overrightarrow{\delta L^n} \cdot \overrightarrow{N^n}$$

donde $\overrightarrow{\delta L^n}$ es la medida de cambio o variación en la información de L^n. La ecuación anterior estima la información de la imagen y calcula su variación a lo largo de la dirección marcada por el vector $\overrightarrow{N^n}$.

Nótese que cuando se llegue a un estado estacionario, $I^{n+1} = I^n$ o $I_t^n = 0$, entonces se tendrá que $\overrightarrow{\delta L^n} \cdot \overrightarrow{N^n} = 0$, lo que significa que L habrá sido propagado de forma correcta en la dirección $\overrightarrow{N}$.

En este punto ya se ha definido el modo de propagación de la información. A partir de aquí, lo que se necesita es conocer la información a propagar L y su dirección de propagación $\overrightarrow{N}$.

El primero de los parámetros que se va a tratar es la información a propagar L. Se quiere que la propagación de la información se realice de forma suave, por lo que L^n deberá se un estimador de suavidad de la imagen. Con este propósito se utilizará una discretización sencilla del *Laplaciano*

$$L^n(i,j) = I_{xx}^n(i,j) + I_{yy}^n(i,j)$$

El principal motivo del uso de una discretización del *Laplaciano* es que este tipo de operador se utiliza en procesado de imagen como un detector de bordes, por tanto, la información representada por este elemento será la existencia de cambios bruscos en la tonalidad de la imagen en el punto.
Con esto y el cálculo de un operador gradiente se conseguirá hallar la variación $\overrightarrow{\delta L^n}$ a lo largo de $\overrightarrow{N}$.

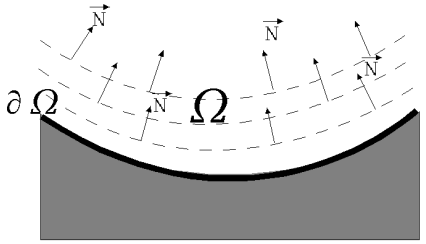


Figura 3.1: Esquema de propagación normal a $\partial\Omega$

El siguiente paso es el de definir la dirección $\overrightarrow{N}$ a lo largo de la cual debe realizarse la propagación.
Para ello se puede pensar que la mejor elección inicialmente se puede definir $\overrightarrow{N}$ como la normal a la distancia a la frontera $\partial\Omega$ con signo, como se muestra en la Figura 3.1. Por ejemplo, para un punto (i,j) en Ω, el vector $\overrightarrow{N}$ en dicho punto será normal a la "versión

contraída" de Ω cuyo sentido irá dirigido hacia el interior de Ω, lo cual desplazará la frontera $\partial\Omega$ a la posición en la cual pertenecerá (i, j). Esta elección se realiza debido a que se piensa que una propagación normal a la frontera $\partial\Omega$ dará lugar a la continuidad de los isófotos en la frontera.
Desafortunadamente, lo que ocurre, como se muestra en la Figura 3.2, es que las líneas de llegada a la frontera $\partial\Omega$ se curvan de modo que tratan de alinearse con $\overrightarrow{N}$, por lo cual, las líneas isófotas tienden a ser perpendiculares a la frontera $\partial\Omega$ y también a sus "versiones contraídas". Esto no es lo que se espera ni tampoco se desea.

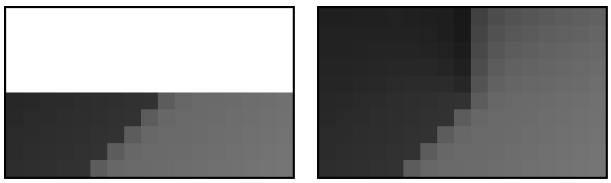


Figura 3.2: Resultado utilizando propagación normal a $\partial\Omega$ (imagen previa e imagen restaurada)

Llegados a este punto, debe hacerse notar que la orientación de $\partial\Omega$ no es intrínseca a la geometría de la imagen, puesto que la región a restaurar es arbitraria.

Una vez conocido que los isófotos tienden a alinearse con $\overrightarrow{N}$, la mejor elección para $\overrightarrow{N}$ será entonces la dirección que marquen los isófotos.

Entonces, se utilizará una estimación de la variación temporal del campo de dirección de los isófotos: para cualquier punto dado (i, j), el vector gradiente $\nabla I^n(i, j)$ (discretizado) dará la dirección de máximo cambio espacial, mientras que su rotación de $\frac{\pi}{2}$ radianes será la dirección de menor cambio espacial, representada por $\nabla^\perp I^n(i, j)$.
Por lo tanto, el vector $\nabla^\perp I^n(i, j)$ dará la dirección de los isófotos. El campo $\overrightarrow{N}$ será de esta forma dado por una dirección variante en el tiempo y definida como

$$\overrightarrow{N}(i, j, n) = \nabla^\perp I^n(i, j)$$

Esta estimación es grosera al inicio del algoritmo, pero de forma progresiva va alcanzando la continuidad deseada en la frontera $\partial\Omega$.

Nótese que el campo de dirección no está normalizado. Para normalizarlo, sencillamente se debe utilizar la norma euclídea del gradiente, $\nabla I(i, j)$.

Todavía existe otra opción más para realizar el cálculo de $\overrightarrow{N}$. Este cálculo se realiza mediante la definición de un tensor de estructura modificado. Para definir este tensor de estructura modificado se empleará un método similar al empleado por Bornemann y März en [11]. Anteriormente, dentro del apartado del esquema semi-implícito del *filtrado de difusión anisótropa*, se ha enunciado la definición de un tensor de estructura clásico siguiendo los pasos de Weickert en [2]. Para comprender los pasos que se realizan en este punto es recomendable comprender primero los calculos clásicos como los enunciados por Weickert en [2], que han sido expuestos anteriormente en el apartado referente al *filtrado de difusión anisótropa*.

El tensor de difusión modificado realiza su cálculo mediante la siguiente fórmula

$$J_\rho(x) = \frac{(K_\rho * (\mathbb{M} \cdot \nabla u_\sigma \otimes \nabla u_\sigma))(x)}{(K_\rho * \mathbb{M})(x)}$$

donde K_ρ es un *kernel gaussiano* de desviación típica $\rho > 0$, $*$ expresa la convolución, $\cdot$ expresa un producto elemento a elemento y ∇u_σ se calcula como

$$u_\sigma = \frac{K_\sigma (\mathbb{M} \cdot I)}{K_\sigma * \mathbb{M}}$$

donde K_σ es un *kernel gaussiano* de desviación típica $\sigma > 0$ y $\mathbb{M}$ es un indicador de la pertenencia del punto a la máscara. $\mathbb{M}$ será una matriz del mismo tamaño que la máscara. En la región Ω los píxeles valdrán 1 ($\mathbb{M}(\Omega) = 1$). Dentro de la imagen, en los píxeles que no pertenezcan a Ω, tendrán un valor muy superior al de la región interna (por ejemplo, $\mathbb{M}(\Omega) = 9$).

Tras el cálculo del tensor de estructura se debe realizar el cálculo de los autovectores como se ha explicado anteriormente en el apartado del esquema semi-implícito del *filtrado de difusión anisótropa*. De esta forma se obviará cualquier estructura que se encuentre en el interior de la región Ω y se obtendrán una continuación de las líneas de los isófotos en el interior de Ω.

Puesto que el *inpainting* se realiza a lo largo de las direcciones indicadas por los isófotos, es irrelevante si $\nabla^\perp I^n(i,j)$ se obtiene girando $\nabla I(i,j)$ a favor o en contra de las agujas del reloj, la dirección determinada será la misma. Lo que sí cambiará dependiendo de la rotación será el sentido del vector, pero eso es irrelevante para este problema.

Recapitulando, se estimará una variación de la suavidad de la imagen en el píxel, dada por una discretización en dos dimensiones del operador Laplaciano, y esta variación será proyectada en la dirección de los isófotos. Esta proyección será utilizada para actualizar el valor de la imagen dentro de la región a ser restaurada.

La ecuación $I_t^n = \overrightarrow{\delta L^n} \cdot \overrightarrow{N^n}$ se puede escribir en forma de EDP (Ecuación en Derivadas Parciales). Con ello se obtiene la *ecuación de inpainting*

$$I_t = \nabla(\Delta I) \cdot \nabla^\perp I$$

Con el objetivo de asegurar una correcta evolución del campo de dirección, se intercalará, con el proceso de inpainting descrito anteriormente, un proceso de difusión. Mediante este proceso de difusión se quiere conseguir dar estabilidad al método y también eliminar el posible ruido que pudiese surgir. Aunque también se podría añadir el proceso de difusión como un término adicional en $I_t^n(i,j)$ obteniendo resultados similares.

Cada A pasos del proceso de inpainting se aplicarán B pasos de difusión, donde $A > B$ y ambos serán números pequeños. Esta difusión corresponde a la curvatura periódica de las líneas para evitar con ello que se crucen las unas con las otras.

Para el proceso de difusión se utilizará difusión anisótropa, para conseguir un proceso sin pérdida de nitidez en la reconstrucción. Dicho proceso se realizará únicamente dentro de la región Ω. Se aplicará una discretización de la siguiente ecuación:

$$\frac{\partial I}{\partial t}(x,y,t) = g_\epsilon(x,y)\,\kappa(x,y,t)\,|\nabla I(x,y,t)|\,, \forall x,y \in \Omega^\epsilon$$

donde

- Ω^ϵ es una dilatación de Ω con una bola de radio ϵ
- $\kappa(x, y, t)$ es la curvatura euclídea de los isófotos en $I(x, y)$
- $g_\epsilon(x, y)$ es una función suavizadora en Ω^ϵ tal que:

$$g_\epsilon(x, y) = \begin{cases} 0 & \text{en} \quad \partial\Omega^\epsilon \\ 1 & \text{en} \quad \Omega \end{cases}$$

  Esta función es una forma de imponer condiciones de frontera de tipo Dirichlet para la ecuación de difusión anterior.

En este punto debe hacerse notar un hecho clave, el *estado estacionario* de la *ecuación de inpainting* se da cuando

$$\nabla(\Delta I) \cdot \nabla^\perp I = 0$$

es el mismo estado que el de la ecuación de Navier-Stokes para la ecuación de flujo de un fluído viscoso incomprensible. En otras palabras, resolver el problema de inpainting para una imagen I es análogo a resolver la ecuación de Navier-Stokes (con condiciones frontera específicas) para un flujo con función de corriente I.

La analogía envuelve otras cantidades también: $\nabla\perp I$ que es la velocidad del fluído, ΔI es la vorticidad, etc. Las implicaciones de este hecho son muy importantes. Entre estas implicaciones se debe destacar que la correspondencia entre las ecuaciones puede ayudar a comprender mejor el proceso de *inpainting* dando lugar a un nuevo enfoque además de solucionar problemas que se pudiesen encontrar.

Capítulo 4

Discretización numérica

El gran objetivo de este trabajo es desarrollar los algoritmos e implementarlos de forma práctica introduciéndolos dentro de una aplicación de un dispositivo móvil. Este capítulo se ha desarrollado con el objetivo de explicar de forma genérica el funcionamiento de los algoritmos que se van a desarrollar para procesar las imágenes.

El motivo principal para realizar una discretización es el paso de una realidad analógica, en la que vivimos, a un mundo digital que es utilizado por los computados y los aparatos electrónicos hoy en día. Como en este mundo digital todo dato es finito y tiene valores en ciertos puntos dados con una precisión máxima preestablecida.
Dentro del mundo de la imagen, existen de ambos tipos. Las imágenes analógicas son aquellas que tomaban las cámaras fotográficas antiguas y quedaban impresas en negativos dentro de películas que después se revelaban en un tipo de papel especial. Matemáticamente se puede decir que dentro del recinto del papel (o recinto de la imagen), la imagen era continua. En cambio, las imágenes digitales son discretizaciones de las imágenes analógicas clásicas.

Una imagen digital se discretiza generalmente como una malla o red de puntos dentro del recinto que comprendería la imagen analógica. A cada uno de los puntos de esa malla se le asignarán uno o varios valores que indicarán el valor de dicho punto. Este valor será dependiente del tipo de formato de color que utilice la imagen digital.
Por otro lado, la malla puede tener más o menos puntos (dependiendo de la distancia que se quiera establecer entre éstos). Cuanto mayor sea el número de puntos, mayor será la resolución, pero también la memoria que la imagen digital ocupe aumentará de forma muy rápida conforme aumente su resolución.

Este capítulo tendrá dos partes bien diferenciadas donde se explicarán las discretizacoines realizadas para los métodos elegidos expuestos en el capítulo anterior. En la primera parte se tratará y expondrá la discretización que se empleará para el *filtrado de difusión anisótropa* y, en la segunda, se realizará lo mismo con el método de *inpainting*.

4.1. Filtrado de difusión anisótropa

Como ya se ha descrito en el capítulo anterior, el filtrado de difusión anisótropa utiliza la ecuación elíptica o ecuación del calor para eliminar de la superficie de la imagen el ruido.

Con esta eliminación de ruido se conseguirá que las imágenes estén más suavizadas en las regiones planas de la imagen y mantengan sus bordes.

4.1.1. Filtro explícito

En esta sección se va a proceder a discretizar la ecuación del calor para el tratamiento de imágenes. Esta discretización va a ser progresiva para que su entendimiento sea más sencillo. La progresión indicada va a pasar inicialmente por describir la discretización para una dimensión y luego, se pasará a formularla para dos dimensiones, que es el caso práctico que finalmente nos ocupará.

También se procederá a analizar la estabilidad de esta solución, que es la propuesta inicialmente por Perona y Malik [1], la cual fue implementada de forma *real* por Gerig [37].

Finalmente, se expondrá la discritización expuesta por Weickert en [2] que es la que se utilizará en la aplicación.
Se ha decidido aplicar la técnica de [2] debido a los buenos resultados que presenta y a la bondad de su estabilidad.

4.1.1.1. Formulación discreta 1D

Para el tratamiento de imágenes es necesario realizar una reformulación en forma discreta de la ecuación de difusión (3.1) definida en el capítulo anterior para los casos contínuos. En este caso, en vez de calcular los gradientes locales, sus valores se pueden aproximar por las diferencias existentes entre los elementos de datos cercanos.
Para proporcionar una mejor apreciación de lo que significa el proceso de difusión no lineal, primero se va a realizar la reformulación para el caso de una dimensión. Para ello, se va a tomar como eje de coordenadas el eje x. Teniendo en cuenta esto, la ecuación de difusión se escribiría del siguiente modo:

$$\frac{\partial I(x,t)}{\partial t} = \operatorname{div}\left(c(x,t)\nabla I(x,t)\right) \tag{4.1}$$

Considerando que para una dimensión: $\operatorname{div} = \nabla = \frac{\partial}{\partial x}$ y sustituyendo en (4.1):

$$\frac{\partial I(x,t)}{\partial t} = \frac{\partial}{\partial x}\left(c(x,t)\frac{\partial I(x,t)}{\partial x}\right) \tag{4.2}$$

Si ahora se tiene en cuenta la aproximación en diferencias:

$$\frac{\partial f(x,t)}{\partial x} \approx \frac{1}{\Delta x}\left[f\left(x+\frac{\Delta x}{2},t\right) - f\left(x-\frac{\Delta x}{2},t\right)\right]$$

y se aplica en (4.2):

$$\frac{\partial I(x,t)}{\partial t} \approx \frac{\partial}{\partial x}\left[c(x,t) * \left(I\left(x+\frac{\Delta x}{2},t\right) - I\left(x-\frac{\Delta x}{2},t\right)\right)\right]$$

Si de nuevo se aplica la aproximación de la diferencial en la anterior ecuación se obtiene:

$$\begin{aligned}&\frac{\partial I(x,t)}{\partial t} \approx \\ &\frac{1}{\Delta x^2}\left[c\left(x+\frac{\Delta x}{2},t\right)\left(I\left(x+\Delta x,t\right)-I\left(x,t\right)\right)\right. \\ &\left.-\,c\left(x-\frac{\Delta x}{2},t\right)\left(I(x,t)-I\left(x-\Delta x,t\right)\right)\right]\end{aligned} \tag{4.3}$$

Esta última ecuación se puede expresar por:

$$\frac{\partial I(x,t)}{\partial t} \approx \Phi_d - \Phi_i \tag{4.4}$$

donde Φ_d representa el flujo local hacia la derecha y Φ_i representa el flujo local hacia la izquierda (véase la Figura (4.1)).

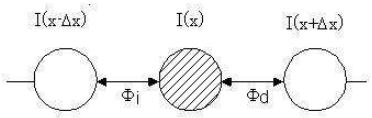


Figura 4.1: Representación de las contribuciones de los flujos existentes para el caso 1D.

Si ahora, de forma similar, en (4.4) se aplica:

$$\frac{\partial f(x,t)}{\partial t} \approx \frac{1}{\Delta t}(f(x,t+\Delta t)-f(x,t))$$

entonces se obtiene:

$$I(x,t+\Delta t) \approx I(x,t) + \Delta t(\Phi_d - \Phi_i) \tag{4.5}$$

Después de haber realizado todo este desarrollo matemático, se puede observar que al discretizar la ecuación de calor se llega a un proceso iterativo. En este proceso iterativo simplemente se indica que el valor de I, para una cierta coordenada x, en el siguiente instante de tiempo (en la siguiente iteración) va a venir dado por el valor de I en este instante de tiempo más un cierto valor que va a depender de la diferencia entre los flujos hacia la izquierda y hacia la derecha existentes entorno al punto x considerado.
La estabilidad de este proceso iterativo, lógicamente, se va a obtener mediante la elección de un valor adecuado para la variación temporal Δt. Como se verá más adelante, Δt no va poder superar cierto valor.

4.1.1.2. Formulación discreta 2D

Una vez visto el desarrollo para el caso de una dimensión, se puede proceder a ver cual sería el resultado que se obtendría en el caso de dos dimensiones. Para ello vamos a tomar como ejes de coordenas los ejes x e y.

A priori, visto el resultado anterior, parece que uno puede predecir fácilmente cual va a ser la expresión final de la ecuación. Pero para evitar que se hagan predicciones erróneas,

veamos como sería el desarrollo. La ecuación de la que se parte, para dos dimensiones, tendría la siguiente expresión:

$$\frac{\partial I(x,y,t)}{\partial t} = \text{div}\left[c(x,y,t)\nabla I(x,y,t)\right]$$

Teniendo en cuenta las aproximaciones de las que se ha hablado para una dimensión y haciendo una extensión de ellas para las dos dimensiones, la anterior ecuación se podría expresar:

$$\begin{aligned}\frac{\partial I(x,y,t)}{\partial t} \approx & \frac{1}{\Delta x^2}\left[c\left(x+\frac{\Delta x}{2},y,t\right)\left(I(x+\Delta x,y,t)-I(x,y,t)\right)\right.\\ & \left.-c\left(x-\frac{\Delta x}{2},y,t\right)\left(I(x,y,t)-I(x-\Delta x,y,t)\right)\right]\\ & +\frac{1}{\Delta y^2}\left[c\left(x,y+\frac{\Delta y}{2},t\right)\left(I(x,y+\Delta y,t)-I(x,y,t)\right)\right.\\ & \left.-c\left(x,y-\frac{\Delta y}{2},t\right)\left(I(x,y,t)-I(x,y-\Delta y,t)\right)\right]\end{aligned} \tag{4.6}$$

Se aprecia que el primer sumando en el término de la derecha de la ecuación (4.6) es similar al término que se tenía a la derecha de la igualdad en (4.3), realmente sólo se diferencia en la presencia de la coordenada y, por tanto este término será equivalente a la diferencia entre dos flujos locales. Uno de ellos será un flujo hacia la derecha y el otro hacia la izquierda en el eje x, al igual que en el caso de una dimensión.
El segundo sumando en (4.6) es igual al primero, salvo que las variaciones se producen en y en vez de en x. Por tanto, también equivaldrá a la diferencia entre dos flujos locales, uno hacia la derecha y el otro hacia la izquierda, pero ahora en el eje y. Viendo esto, entonces, se puede ya dar la expresión final de la discretización para el caso en dos dimensiones:

$$I(x,y,t+\Delta t) \approx I(x,y,t) + \Delta t(\Phi_e - \Phi_w + \Phi_n - \Phi_s) \tag{4.7}$$

donde Φ_e y Φ_w son los flujos hacia el este y hacia el oeste, que se corresponderían con los flujos hacia la derecha e izquierda en el eje x y Φ_n y Φ_s son los flujos norte y sur que se corresponden con los flujos hacia la derecha e izquierda en el eje y.
Como en el caso de una dimensión se obtiene un proceso iterativo en el que en cada nuevo paso temporal $t+\Delta t$ se genera una nueva imagen a partir de la imagen del paso temporal t más un cierto valor que va a depender de las diferencias entre los flujos locales. Lógicamente la estabilidad del resultado va a venir determinada por la variación temporal entre iteraciones que representa el parámetro Δt.

4.1.1.3. Aplicación a imágenes

Puesto que lo que interesa es poder aplicar la ecuación (4.7) a imágenes se debe expresar dicha ecuación en función de los píxeles de la imagen. $I(x,y,t)$, para un cierto valor de x y otro de y, hará referencia a la intensidad de un cierto pixel de la imagen en un instante de tiempo. El valor de éste en el siguiente paso temporal, como se ha dicho ya varias veces, estará influenciado por los cuatro vecinos situados al norte, sur, este y oeste. Pero para obtener mejores resultados, a la hora de calcular los flujos, no sólo se van a tener en cuenta

estos cuatro píxeles, sino que también se consideraran los cuatros vecinos diagonales. Por tanto, se va a trabajar con una matriz de 3×3 centrada en el pixel de estudio. Un ejemplo claro de esta región de estudio se puede observar en la Figura (4.2), donde P5 es el píxel central del cual se calculará el valor.

P1	P2	P3
P4	P5	P6
P7	P8	P9

Figura 4.2: Vecindario 3x3.

Las expresiones de los cuatro flujos, que se deben de poner en función de los píxeles, son las siguientes:

$$\begin{aligned}
\Phi_e &= \frac{1}{\Delta x^2}\left[c(x+\frac{\Delta x}{2},y,t)\,(I(x+\Delta x,y,t)-I(x,y,t))\right] \\
\Phi_w &= \frac{1}{\Delta x^2}\left[c(x-\frac{\Delta x}{2},y,t)\,(I(x,y,t)-I(x-\Delta x,y,t))\right] \\
\Phi_n &= \frac{1}{\Delta y^2}\left[c(x,y+\frac{\Delta y}{2},t)\,(I(x,y+\Delta y,t)-I(x,y,t))\right] \\
\Phi_s &= \frac{1}{\Delta y^2}\left[c(x,y-\frac{\Delta y}{2},t)\,(I(x,y,t)-I(x,y-\Delta y,t))\right]
\end{aligned} \tag{4.8}$$

Además el *coeficiente de difusión*, siguiendo la recomendación de Perona y Malik, se expresa como una función del *gradiente*:

$$c(x,y,t) = f(\|\nabla I(x,y,t)\|)$$

Y si se hace la aproximación del *gradiente* vista anteriormente, se tiene para un cierto t:

$$\begin{aligned}
\|\nabla I(x,y)\| \approx \Bigg[&\frac{1}{\Delta x^2}\left[I\left(x+\frac{\Delta x}{2},y\right)-I\left(x-\frac{\Delta x}{2},y\right)\right]^2 \\
&+\frac{1}{\Delta y^2}\left[I\left(x,y+\frac{\Delta y}{2}\right)-I\left(x,y-\frac{\Delta y}{2}\right)\right]^2\Bigg]^{\frac{1}{2}}
\end{aligned}$$

Tras las operaciones pertinentes, se pueden obtener los siguientes *gradientes* en función

de los píxeles de la matriz centrada en P_5:

$$\begin{aligned}
\left\|\nabla I\left(x+\frac{\Delta x}{2},y\right)\right\| &\approx \sqrt{\frac{1}{\Delta x^2}\left(P_6-P_5\right)^2+\frac{1}{\Delta y^2}\left(\frac{P_9-P_3}{2}\right)^2}\\
\left\|\nabla I\left(x-\frac{\Delta x}{2},y\right)\right\| &\approx \sqrt{\frac{1}{\Delta x^2}\left(P_5-P_4\right)^2+\frac{1}{\Delta y^2}\left(\frac{P_7-P_1}{2}\right)^2}\\
\left\|\nabla I\left(x,y+\frac{\Delta y}{2}\right)\right\| &\approx \sqrt{\frac{1}{\Delta x^2}\left(\frac{P_9-P_7}{2}\right)^2+\frac{1}{\Delta y^2}\left(P_8-P_5\right)^2}\\
\left\|\nabla I\left(x,y-\frac{\Delta y}{2}\right)\right\| &\approx \sqrt{\frac{1}{\Delta x^2}\left(\frac{P_3-P_1}{2}\right)^2+\frac{1}{\Delta y^2}\left(P_5-P_2\right)^2}
\end{aligned} \tag{4.9}$$

Estas expresiones permiten ya expresar los flujos en función de los píxeles:

$$\begin{aligned}
\Phi_e &= \Phi_e(P_6,P_5,P_9,P_3)\\
\Phi_w &= \Phi_w(P_5,P_4,P_7,P_1)\\
\Phi_n &= \Phi_n(P_8,P_5,P_9,P_7)\\
\Phi_s &= \Phi_s(P_5,P_2,P_3,P_1)
\end{aligned} \tag{4.10}$$

Se representan de forma esquemática cuales son los píxeles que intervienen en cada flujo en la Figura (4.3). Se aprecia, como ya se dijo antes y se indica en (4.10), que para calcular el flujo local en una determinada dirección se tienen en cuenta además los dos píxeles vecinos en las diagonales. Así, por ejemplo, para calcular el flujo local hacia el norte no sólo se tienen en cuenta P_8 y P_5, sino que también P_9 y P_7.
Un hecho importante que hay que considerar es que la distancia existente con los vecinos diagonales lógicamente es mayor. Para tenerlo en cuenta en el esquema es necesario hacer la siguiente asignación: $\Delta x = \Delta = \sqrt{2}$.

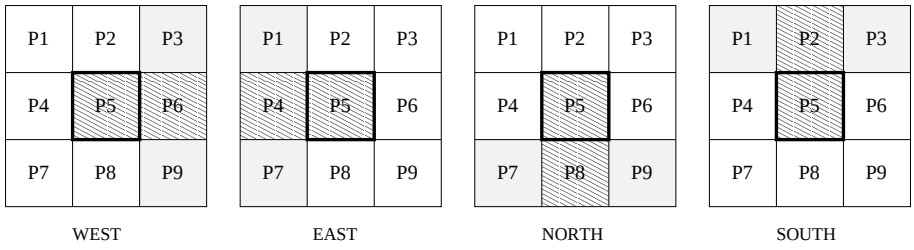


Figura 4.3: Píxeles que intervienen en los distintos flujos locales.

La experiencia ha mostrado que no es necesario buscar esquemas más complicados que incluyan más puntos, con los que se consideran (Figura (4.2)) ya se obtienen buenos resultados.

4.1.1.4. Cálculo de Δt

Tanto en el caso de formulación discreta de la ecuación de difusión para una dimensión (4.4) como en las de dos dimensiones (4.7) y tres dimensiones aparece el factor Δt, que como se sabe representa el tamaño del paso temporal, del que sólo se ha dicho que su valor

debe ser escogido correctamente para evitar llegar a resultados inestables, pero no se le ha asignado ningún valor en concreto. En este apartado se va a proceder a realizar un estudio sobre los posibles valores para su elección.

Cuanto menor sea el valor de esta constante de integración la estabilidad del proceso iterativo será mayor. Pero escoger un valor muy pequeño requiere un elevado número de iteraciones. Por tanto, parece importante ver cual es el límite superior de Δt, para así, aproximándose a ese límite, se evitarán inestabilidades y elevadas iteraciones. Para calcular dicho límite, lógicamente, se parte de la formulación discreta del proceso de difusión, teniendo sólo en cuenta la variable temporal t:

$$I(t + \Delta t) \approx I(t) + \Delta t \; I_t$$

donde I_t representa $\frac{\partial I}{\partial t}$. Y si se sustituye I_t por su valor se obtiene:

$$I(t + \Delta t) \approx I_0 + \Delta t \sum_{i=1}^{n} c_i \; (I_i - I_0) = I_0 \left(1 - \Delta t \sum_{i=1}^{n} c_i\right) + \Delta t \sum_{i=1}^{n} c_i \; I_i$$

donde I_i representan las intensidades vecinas a I_0, que es la que se tiene como referencia para hacer los cálculos, n representa el número de vecinos considerados al calcular los flujos locales para cada caso y c_i son los *coeficientes de difusión*.
Para alcanzar una variación monótona de las intensidades el peso de I_0 debe ser mayor o igual que los pesos de los vecinos I_i:

$$1 - \Delta t \sum_{i=1}^{n} c_i \geq \Delta t \; c_k, \qquad k \in \{1, \ldots, n\} \tag{4.11}$$

En el caso de máxima difusión los coeficientes de difusión son iguales a 1 ($c_k = 1$). Si se tiene en cuenta esto y desarrollando (4.11) se obtiene el límite superior de Δt:

$$\Delta t \; (c_k + \sum_{i=1}^{n} c_i) \leq 1, \quad c_i = 1, \; i \in \{1, \ldots, n\}$$

$$\Delta t \leq \frac{1}{1+n}$$

Si se consideran las diferentes posibilidades de n se obtiene la Tabla 4.1.

Dimensiones	**núm. vecinos**	$1+n$	**máximo** Δt
1D	2	3	1/3
2D	4	5	1/5
	8	9	1/9
3D	6	7	1/7
	26	47/3	3/47

Cuadro 4.1: Máximo valor de Δt para las diferentes dimensiones y número de vecinos

Para aclarar posibles dudas acerca de los resultados mostrados en la Tabla 4.1 hay que decir que en los cálculos realizados anteriormente se consideraba $\Delta x^2 = 1$, esto es debido

a que no se tienen en cuenta los vecinos situados en las diagonales, etc. En el caso de considerar un mayor número de vecinos este factor debe ir tomando otros valores. Así, por ejemplo, para para dos dimensiones, si se consideran 8 vecinos, el máximo Δt es $1/7$ (si se considerase $\Delta x^2 = 1$ debía de ser $1/9$), esto es debido a que en este caso se tiene:

$$\sum_{i=1}^{8} c_i = 4 \cdot 1 + 4 \cdot \frac{1}{2} = 6$$

$$\Delta t \leq \frac{1}{1+6} = \frac{1}{7}$$

Existen 4 vecinos con $c_i = 1$ y $\Delta x^2 = 1$ y otros 4 vecinos, los de las diagonales, para los que hay que considerar que $\frac{1}{\Delta x^2} = \frac{1}{2}$.
Análogamente, en el caso tres dimensiones con 26 vecinos se tienen 6 vecinos con $\frac{1}{\Delta x^2} = 1$, 12 en las diagonales con $\frac{1}{\Delta x^2} = \frac{1}{2}$ y 8 en las esquinas del cubo con $\frac{1}{\Delta x^2} = \frac{1}{3}$:

$$\sum_{i=1}^{2} 6c_i = 6 \cdot 1 + 12 \cdot \frac{1}{2} + 8 \cdot \frac{1}{3} = 12 + \frac{8}{3} = \frac{44}{3} \Delta t \leq \frac{1}{1 + \frac{44}{3}} = \frac{3}{47}$$

4.1.2. Filtro semi-implícito

La gran diferencia entre Weickert y el filtrado de Perona-Malik se centra en el carácter puramente *anisótropo* del filtro de Weickert. Como se puede observar en el método anterior, para calcular el *coeficiente de difusión* se utiliza $\|\nabla I(x,y)\|$, pero en este caso se utiliza un descriptor que puede utilizar de forma independiente los autovalores correspondientes a los autovectores por separado.

Por otro lado, en el capítulo anterior se ha introducido ya el problema continuo P_c mediante el cual se realiza la difusión y también se ha introducido el cálculo del tensor de difusión que se empleará. Este cálculo se ha realizado de

4.1.2.1. Discretización de la imagen de partida

Una imagen discreta puede ser vista como un vector $f \in \mathbb{R}^N, N \geq 2$, cuyas componentes $f_j, j = 1, ..., N$ representan los valores de gris en cada píxel. De ahora en adelante, se denotará el conjunto de índices $1, ..., N$ como J.

Para especificar los requisitos del filtro es necesaria la definición de matriz irreducible:
Definición: Una matriz $A = (a_{ij}) \in \mathbb{R}^{N \times N}$ es definida como **irreducible** si para cualquier $i, j \in J$ existe $k_0, ..., k_r \in J$ con $k_0 = i$ y $k_r = j$ tal que $a_{k_p k_{p+1}} \neq 0$ para $p = 0, ..., r-1$.

Con ello el problema a tratar (P_s) se define de la forma siguiente:

Sea $f \in \mathbb{R}^N$, encontrar la funcion $u \in C^1([0,\infty), \mathbb{R}^N)$ que satisface un problema de valores iniciales del tipo:

$$\begin{aligned} \frac{\partial u}{\partial t} &= A(u)u \\ u(0) &= f \end{aligned}$$

donde $A = (a_{ij})$ cumple las siguientes propiedades

1. Continuidad de tipo Lipschitz de $A \in C(\mathbb{R}^N, \mathbb{R}^{N\times N})$ para todo subconjunto acotado de $\mathbb{R}^N$.
2. Simetría
   $a_{ij}(u) = a_{ji}(u) \;\; \forall i, j \in J, \; \forall u \in \mathbb{R}^N$
3. Equilibrio en las componentes de las filas
   $\sum_{j\in J} a_{ij}(u) = 0 \;\; \forall i \in J, \; \forall u \in \mathbb{R}^N$
4. Elementos no negativos fuera de la diagonal principal
   $a_{ij}(u) \geq 0 \;\; \forall i \neq j \in J, \; \forall u \in \mathbb{R}^N$
5. Irreducibilidad $\forall u \in \mathbb{R}^N$

Estos requisitos son necesarios para

- Tener un problema "bien puesto", al igual que (P_c).
- La solución está acotada.
- El mantenimiento del nivel medio de gris dentro de Ω a lo largo de la evolución temporal.
- Existencia de funciones de Lyapunov que demuestran la estabilidad del sistema.
- Convergencia de la solución a un estado estacionario.

La discretización del problema se realizará de la siguiente forma.
Sea el rectángulo $\Omega = (0, a_1) \times (0, a_2)$ discretizado por una malla de $N = m \times n$ píxeles tal que cada píxel (i, j) con $1 \leq i \leq m$ y $1 \leq j \leq n$ representa la localización (x_i, x_j) donde

$$x_i = \left(i - \frac{1}{2}\right) \; h_1;$$
$$y_j = \left(j - \frac{1}{2}\right) \; h_2;$$

y los tamaños de separación de la malla h_1 y h_2 vienen dados por

$$h_1 = \frac{a_1}{m}$$
$$h_2 = \frac{a_2}{n}$$

En el caso que se trata, el de imágenes digitales, estos dos valores h_1 y h_2 tendrán valor 1, ya que los píxeles están equiespaciados en las dos direcciones de la malla y presentados en una malla regular.

Estos píxeles pueden ser representados mediante una función biyectiva

$$p : \{1, 2, \ldots, m\} \times \{1, 2, \ldots, n\} \to \{1, 2, \ldots, N\}$$

con lo que el píxel (i, j) puede ser representado como un único índice $p(i, j)$.

Esta discretización cumple con las cinco propiedades de (P_s) tanto para el caso isótropo como para el anisótropo, lo cual hace posible el uso de discretizaciones espaciales del problema (P_c).

En el caso isótropo la demostración de las propiedades es sencilla, pero en el anisótropo la comprobación de la cuarta propiedad (elementos fuera de la diagonal principal no nulos) es más compleja y se establece mediante el siguiente teorema:

Teorema 4.1.1. *Sea $D \in \mathbb{R}^{2\times 2}$ una matriz simétrica definida positiva con un radio espectal κ. Entonces existe $m(\kappa) \in \mathbb{N}$ tal que $div(D\nabla u)$ da lugar a una discretización en diferencias finitas en una región $(2m+1)\times(2m+1)$ alrededor del punto.*

La discretización espacial se puede aplicar de forma práctica a una región 3×3 utilizando

$$D = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$$

donde a, b y c son funciones de $J_\rho(\nabla u_\sigma)$ se obtiene la discretización de segundo orden que se muestra en la figura 4.4. En dicha figura donde están representadas todas las entradas no nulas de la p-ésima fila de $A(u)$. El punto $p(i,j)$ es un índice que representa a un píxel interior. Los subíndices indican el valor en la posición (x_i, y_j) de las funciones $a(J_\rho(\nabla u_\sigma))$, $b(J_\rho(\nabla u_\sigma))$ y $c(J_\rho(\nabla u_\sigma))$.

4.1.2.2. Tensor de difusión

En el capítulo anterior ya se ha comentado el cálculo del tensor de difusión que se utilizará en el método y se ha especificado casi todo su cálculo. En este apartado se hará incapié en el cálculo discreto del tensor.

Inicialmente el tensor de estructura se ha definido en la ecuación (3.8), la cual se utilizará para calcular el tensor de estructura en cada uno de los puntos de la imagen.
La ecuación (3.8) se construía partiendo de la imagen inicial u, la cual se filtraba mediante un *kernel gaussiano* K_σ de desviación típica σ para hacer el tensor de estructura inmune al ruido que pueda presentar la imagen. El resultado de esta imagen filtrada se denotaba u_σ de la cual se calculaba el gradiente.
Siguiendo los pasos de Scharr en [44], dicho gradiente se obtendrá aplicando los *kernels* que se muestran la Figura 4.5.

Tras esto se procederá a realizar el cálculo de los autovalores y autovectores para cada uno de los puntos según se indica en el capítulo anterior y posteriormente se realizará el cómputo del tensor de difusión que se utilizará para filtrar la imagen y será aplicado en el esquema representado en la Figura (4.4).

$\frac{\lvert b_{i-1,j+1}\rvert - b_{i-1,j+1}}{4h_1h_2}$ $+\frac{\lvert b_{i,j}\rvert - b_{i,j}}{4h_1h_2}$	$\frac{c_{i,j+1}+c_{i,j}}{2h_2^2} - \frac{\lvert b_{i,j+1}\rvert + \lvert b_{i,j}\rvert}{2h_1h_2}$	$\frac{\lvert b_{i+1,j+1}\rvert + b_{i+1,j+1}}{4h_1h_2}$ $+\frac{\lvert b_{i,j}\rvert - b_{i,j}}{4h_1h_2}$
$\frac{a_{i-1,j}+a_{i,j}}{2h_1^2}$ $-\frac{\lvert b_{i-1,j}\rvert + \lvert b_{i,j}\rvert}{2h_1h_2}$	$-\frac{a_{i-1,j}+2a_{i,j}+a_{i+1,j}}{2h_1^2}$ $-\frac{\lvert b_{i-1,j+1}\rvert - b_{i-1,j+1} + \lvert b_{i+1,j+1}\rvert + b_{i+1,j+1}}{4h_1h_2}$ $-\frac{\lvert b_{i-1,j-1}\rvert + b_{i-1,j-1} + \lvert b_{i+1,j-1}\rvert - b_{i+1,j+1}}{4h_1h_2}$ $+\frac{\lvert b_{i-1,j}\rvert + \lvert b_{i+1,j}\rvert + \lvert b_{i,j-1}\rvert + \lvert b_{i,j+1}\rvert + 2\lvert b_{i,j}\rvert}{2h_1h_2}$ $-\frac{c_{i,j-1}+2c_{i,j}+c_{i,j+1}}{2h_2^2}$	$\frac{a_{i+1,j}+a_{i,j}}{2h_1^2}$ $-\frac{\lvert b_{i+1,j}\rvert + \lvert b_{i,j}\rvert}{2h_1h_2}$
$\frac{\lvert b_{i-1,j+1}\rvert - b_{i-1,j+1}}{4h_1h_2}$ $+\frac{\lvert b_{i,j}\rvert - b_{i,j}}{4h_1h_2}$	$\frac{c_{i,j+1}+c_{i,j}}{2h_2^2} - \frac{\lvert b_{i,j+1}\rvert + \lvert b_{i,j}\rvert}{2h_1h_2}$	$\frac{\lvert b_{i+1,j+1}\rvert + b_{i+1,j+1}}{4h_1h_2}$ $+\frac{\lvert b_{i,j}\rvert - b_{i,j}}{4h_1h_2}$

Figura 4.4: Discretización de segundo orden de una región 3×3

$$\partial_x = \frac{1}{32}\begin{array}{|c|c|c|}\hline 3 & 0 & -3 \\ \hline 10 & 0 & -10 \\ \hline 3 & 0 & -3 \\ \hline\end{array} \qquad \partial_y = \frac{1}{32}\begin{array}{|c|c|c|}\hline -3 & -10 & -3 \\ \hline 0 & 0 & 0 \\ \hline 3 & 10 & 3 \\ \hline\end{array}$$

Figura 4.5: *Kernels* Para el cálculo del gradiente.

4.1.2.3. Discretización de la ecuación

A este tipo de filtro se le pueden aplicar discretizaciones de tipo semi-implícito del problema (P_s). Para ello se puede actuar derivando (P_s) para obtener (P_d).

Denotando de igual forma que en los casos anteriores la imagen, como $f \in \mathbb{R}^N, N \geq 2$, y el conjunto de índices, $\{1, \cdots, N\}$ como J, se considera (P_d) de la siguiente forma

Calcular la secuencia $(u^{(k)})_{k\in\mathbb{N}_0}$ de versiones sucesivas de la imagen inicial f como medias de

$$\begin{aligned} u^{(0)} &= f \\ u^{(k+1)} &= Q(u^{(k)})u^{(k)},\ \forall k \in \mathbb{N}_0 \end{aligned}$$

donde $Q = (q_{ij})$ cumple las siguientes propiedades

1. Continuidad en su argumento
   $Q \in C(\mathbb{R}^N, \mathbb{R}^{N\times N})$
2. Simetría
   $q_{ij}(v) = q_{ji}(v)\ \ \forall i, j \in J,\ \forall v \in \mathbb{R}^N$
3. Suma unitaria en las filas
   $\sum_{j\in J} q_{ij}(v) = 1\ \ \forall i, j \in J,\ \forall v \in \mathbb{R}^N$
4. No negatividad
   $q_{ij}(v) \geq 0\ \ \forall i \neq j \in J,\ \forall v \in \mathbb{R}^N$
5. Irreducibilidad $\forall v \in \mathbb{R}^N$
6. Elementos diagonales positivos
   $q_{ij}(v) > 0\ \ \forall i \in J,\ \forall v \in \mathbb{R}^N$

Estas propiedades de $Q(u^{(k)})$, al igual que en el caso semidiscreto, se convierten en una serie de requisitos necesarios para que se cumpla que

- El problema esté "bien puesto".
- La solución sea regular y se mantenga acotada.
- Se mantenga el valor medio de gris en la imagen.
- La existencia de secuencias de Lyapunov para garantizar la estabilidad del método.

- La convergencia de la solución a un estado estacionario

Para estudiar la estabilidad del problema se partirá de que $u^{(k)}$ es la aproximación de la solución u de (P_s) en un tiempo $t = k$ dt donde dt es el paso temporal por iteración. El esquema en diferencias finitas será semi-implícito dependiente de un parámetro α. Dentro de este esquema A será un operador que dependerá de forma no lineal de u.

La estabilidad viene enunciada por el siguiente Teorema:

Teorema 4.1.2. *Sea $\alpha \in [0,1]$, dt > 0 y $A = (a_{ij}) : \mathbb{R}^N \to \mathbb{R}^{N\times N}$ que satisface las propiedades de (P_s). Entonces el esquema semi-implícito*

$$\frac{u^{(k+1)} - u^{(k)}}{\mathrm{dt}} = A(u^{(k)})(\alpha u^{(k+1)} + (1-\alpha)u^{(k)})$$

cumple todas las propiedades de (P_d) para los modelos discretos de difusión siempre que

$$\mathrm{dt} \leq \frac{1}{(1-\alpha)\max\limits_{i\in J}\left|a_{ii}u^(k)\right|}$$

para $\alpha \in (0,1)$.
En el caso explícito ($\alpha = 0$), las propiedades de (P_d) se cumplen en el caso de que

$$\mathrm{dt} \leq \frac{1}{\max\limits_{i\in J}\left|a_{ii}u^(k)\right|}$$

Para el caso semi-implícito ($\alpha = 1$), estas propiedades se cumplen siempre.

En este caso se va a considerar el esquema semi-implícito

$$\frac{u^{(k+1)} - u^{(k)}}{\mathrm{dt}} = A(u^{(k)})u^{(k+1)}$$

lo cual lleva a

$$\left[I - \ \mathrm{dt}\ A(u^{(k)})\right] u^{(k+1)} = u^{(k)}$$

Este sistema no aporta la solución de forma directa, requiere en primer lugar resolver un sitema lineal, esa es la razón por la que se le denomina semi-implícito.

Este sistema ya se podrá resolver mediante métodos sencillos y muy tratados en el cálculo numérico, como puede ser el algoritmo de *Thomas* para sistemas tridiagonales, como enuncia en [3].

También Weickert nombra en su libro [2] y en su artículo [3], que es posible aplicar otros esquemas denominados AOS, de sus siglas en inglés *Additive Operator Splitting*, en castellano *descomposicion en operadores aditivos.*

Este esquema *AOS* reside en su fundamento en que resolver el sistema lineal con la matriz $\left[I - \ \mathrm{dt}\ A(u^{(k)})\right]$ puede ser computacionalmente costoso. Por ello, se busca una solución más eficiente. Esta solución consiste en que se conoce una forma de descomponer la matriz $A(u^{(k)})$ de la forma siguiente

$$A(u^{(k)}) = \sum_{l=1}^{m} A_l(u^{(k)})$$

Mediante esta descomposición se tendrán m sistemas lineales y sus matrices serán $\left[I - m\alpha \text{ dt } A_l(u^{(k)})\right], l = 1, \ldots, m$ los cuales podrán ser resueltos de una forma más eficiente.

Con esta formulación lo que se tratará de estudiar será el siguiente sistema

$$u^{(k+1)} = \left[I - \alpha \text{ dt} \sum_{l=1}^{m} A_l(u^{(k)})\right]^{-1} \left[I + (1-\alpha) \text{ dt} \sum_{l=1}^{m} A_l(u^{(k)})\right] u^{(k)}$$

cuya variante AOS es

$$u^{(k+1)} = \frac{1}{m} \sum_{l=1}^{m} \left[I - \alpha m \text{ dt } A_l(u^{(k)})\right]^{-1} \left[I + (1-\alpha) m \text{ dt } A_l(u^{(k)})\right] u^{(k)}$$

La estabilidad de estos esquemas viene dada por el siguiente teorema

Teorema 4.1.3. *Sea $\alpha \in (0,1]$, dt > 0 y sea $A_l = (a_{ijl})_{ij} : \mathbb{R}^N \to \mathbb{R}^{N \times N}$, $l = 1, \ldots, m$ que satisface las propiedades de la matriz $A(u)$ introducida en (P_s).*
Además, si se asume que $A(u) = \sum_{l=1}^{m} A_l(u)$ es irreducible $\forall u \in \mathbb{R}^N$ y que por cada A_l existe una permutación $P_l \in \mathbb{R}^{N \times N}$ tal que $P_l A_l P_l^T$ es diagonal por bloques y e irreducible dentro del cada bloque. Entonces, para $\alpha \in (0,1)$, el esquema AOS cumple con las propiedades de (P_d) para la difusión discreta en el caso de que

$$\text{dt} < \frac{1}{(1-\alpha)\, m \max\limits_{i,l} \left|a_i il(u^{(k)})\right|}$$

En el caso semi-implícito ($\alpha = 1$) las propiedades de (P_d) se satisfacen de forma incondicional.

Entonces en el caso semi-implícito, que será el utilizado finalmente, la formulación del esquema AOS utilizado será

$$u^{(k+1)} = \frac{1}{m} \sum_{l=1}^{m} \left[I - m \text{ dt } A_l(u^{(k)})\right]^{-1} u^{(k)}$$

4.1.3. Imágenes a color

Las imágenes a color son diferentes a las imágenes en escala de grises. Una imagen en escala de grises puede verse como una aplicación que se da en el interior de una región $\Omega := (0, a_1) \times (0, a_2)$. Esta aplicación se definirá de la siguiente forma

$$\text{Imagen escala grises} : \Omega \to \mathbb{R}$$

En cambio, en el caso de una imagen a color, ésta está compuesta por tres canales no sólo uno, por tanto la aplicación deberá definirse como

$$\text{Imagen a color} : \Omega \to \mathbb{R}^3$$

En el caso de aplicar este esquema en imágenes en color, se puede realizar de varias formas.

La primera de ellas sería tomar cada unos de los canales de color de la imagen como una imagen y aplicar el esquema de forma separada a todos ellos. Esto podría provocar que en algunos lugares pudieran llegar a perderse estructuras e incluso que apareciesen colores espúreos, dando lugar a regiones dentro de las imágenes con mayor nivel de ruido que la inicial.

Otra forma es la utilizada en [4]. En este artículo se enuncia que partiendo de una imagen con múltiples canales $u_i, i = 1, \ldots, m$ se pueden considerar los autovectores de

$$\sum_{i=1}^{m} \nabla u_i (\nabla u_i)^T$$

Esta idea es extensible y, mediante la introducción de ruido y la escala de integración, se puede conseguir un tensor de estructura común a todos los canales añadiendo la aportación de cada uno de los canales de la siguiente forma

$$J_\rho(\nabla \vec{u}_\sigma) := \sum_{i=1}^{m} J_\rho(\nabla \vec{u}_{i,\sigma})$$

En vez de eso se ha decidido realizar un procedimiento distinto. Este procedimiento esta basado en realizar el cómputo del tensor de difusión de forma conjunta para toda la imagen. Para obtener dicho tensor de difusión se transformará la imagen de formato RGB (acrónimo de Red (Rojo), Green (Verde), Blue (Azul)) a una imagen en niveles de gris mediante la siguiente ecuación:

$$GS(i,j) = 0{,}2989 \cdot R(i,j) + 0{,}5870 \cdot G(i,j) + 0{,}1140 \cdot B(i,j)$$

donde $GS(i,j)$ será el nivel de gris de la imagen en el píxel (i,j).

En esta imagen en escala de grises se calcularán el tensor de difusión para cada uno de los puntos. Con ello se conseguirá un algoritmo más rápido en el sentido computacional ya que será necesario calcular el tensor una única vez para toda la imagen en vez de tres como se proponía en el primer método, aunque como contrapartida de ello es necesario realizar. Tras el cómputo del tensor de difusión se realizará la difusión y el avance de la ecuación de difusión en la imagen. Esta difusión será realizada en cada uno de los tres canales de color de forma independiente.

4.2. *Inpainting*

Dentro de la sección *inpainting* que se puede encontrar en el capítulo anterior ya se ha descrito la combinación de métodos que se iba a utilizar. Estos métodos no han sido descritos a fondo debido a que no han sido discretizados. En esta sección se encuentran los detalles completos que se han utilizado para la implementación del método.

4.2.1. Esquema discreto

Para el algoritmo únicamente serán necesarias dos entradas:

- Imagen a restaurar.
- Máscara indicadora de la región a restaurar que genéricamente puede ser binaria indicando o no la pertenencia de los píxeles.

4.2.1.1. Aproximación rápida del resultado

Como ya se ha nombrado antes, se realizará una aproximación grosera a la solución. Para realizarla se implementará un algoritmo de tipo *onion-peel* para ir restaurando los píxeles desde la frontera $\partial\Omega$ de la máscara hacia su interior.

Para localizar los puntos pertenecientes a la frontera $\partial\Omega$ de la máscara se utilizará un detector de bordes sencillo como es el laplaciano discreto como el de la Figura 4.6.
El laplaciano será aplicado sobre la matriz M en la que se almacena la máscara donde se indica la región Ω^0. Dentro de esta matriz M, la máscara tendrá valor 1 en aquellos píxeles que pertenezcan a Ω^0 y su valor será 0 en aquellos valores donde no lo sea.
Al aplicar el laplaciano discreto, se obtendrán los bordes de la máscara. Aquellos píxeles que se encuentren en zonas planas tendrán valor 0, los que se encuentren fuera de Ω^0 pero junto a su frontera tendrán un valor positivo ($\Delta M(x,y) > 0$, $x, y \notin \Omega$) y los que se encuentren dentro de Ω^0 junto a la frontera, tendrán valor negativo ($\Delta M(x,y) < 0$, $x, y \in \Omega$).
Estos píxeles con valor del laplaciano negativo serán los que se restauren en la primera iteración ya que forman la frontera de la región a restaurar $\partial\Omega^0$.
Tras la restauración de los píxeles se realizará una contracción de la región Ω^0 eliminando de ella los píxeles que ya no pertenezcan a ella. O, dicho con otras palabras, que acaben de ser restaurados.

0	1	0
1	-4	1
0	1	0

Figura 4.6: *Kernel* para el cálculo del laplaciano discreto.

Como ya se ha comentado anteriormente, estos pasos de localización, restauración y contracción de la frontera se realizarán de forma iterativa hasta que $\Omega = \{\emptyset\}$.

Una vez resuelto el orden de la restauración de los píxeles, el siguiente es la restauración de cada uno de éstos. Para restaurar cada píxel se tendrá en cuenta únicamente aquellos píxeles que no pertenezcan a Ω, es decir, que tengan ya asignado un valor correcto. Con este objetivo se utilizará la ecuación (3.9).

En el caso de la implementación que se va a realizar se utilizará un vecindario 3×3 para que el cómputo necesario sea mínimo. Como función de peso se utilizará el siguiente cálculo

$$w(x,y) = \frac{1 - M(x)}{\|x - y\|}$$

Se recomienda que el vecindario utilizado en el algoritmo no sea muy grande, ya que cuanto mayor sea la dimensión, más se podrán emborronar o distorsionar los bordes que se introduzcan en la imagen. Otro detalle que también se debe tener en cuenta en cuanto

al tamaño del vecindario, es que cuanto mayor sea, mayor será el número de operaciones y, por tanto, tiempo necesario para calcular el valor de cada píxel.

4.2.1.2. Refinado y corrección de la solución

Como un procesado inicial Bertamío en [6] indica que se le puede aplicar un suavizado de difusión anisótropa a la imagen completa. El propósito de este suavizado es el de reducir cualquier afección del ruido sobre el cálculo de las direcciones de los isófotos.

Tras este suavizado dará comienzo el bucle de *inpainting*, donde sólo se realizarán modificaciones de los valores de la imagen que se encuentren dentro de la región Ω. Estos valores cambiarán de acuerdo con la implementación del procedimiento discreto de *inpainting*.

Cada pocas iteraciones de *inpainting*, se aplicará uno o dos pasos de difusión anisótropa. Dicho proceso será repetido hasta que se consiga alcanzar un estado estacionario o se alcance un número máximo de iteraciones preestablecido.

Dentro del algoritmo existirán una serie de parámetros que han sido descritos anteriormente, indicando el número de iteraciones y el avance en cada iteración de cada uno de los procesos que entraña el método de refinado.

A continuación se va a definir formalmente el algoritmo discreto utilizado.

Se comienza denominando $I^n(i,j)$ como uno de los píxeles de la región Ω en un tiempo de *inpainting* que denominaremos n y corresponderá con el número de iteración. Entonces, la *ecuación discreta de inpainting* vendrá dada por

$$I^{n+1}(i,j) = I^n(i,j) + \Delta t\, I_t^n(i,j), \forall (i,j) \in \Omega \tag{4.12}$$

donde

$$I_t^n(i,j) = \left(\overrightarrow{\delta L^n}(i,j) \cdot \frac{\overrightarrow{N}(i,j,n)}{\left\| \overrightarrow{N}(i,j,n) \right\|} \right) \|\nabla I^n(i,j)\| \tag{4.13}$$

con

$$\overrightarrow{\delta L^n}(i,j) := \begin{pmatrix} L^n(i+1,j) - L^n(i-1,j) \\ L^n(i,j+1) - L^n(i,j-1) \end{pmatrix} \tag{4.14}$$

$$L^n(i,j) = I_{xx}^n(i,j) + I_{yy}^n(i,j) \tag{4.15}$$

$$\overrightarrow{N^n}(i,j) := \begin{pmatrix} -I_y^n(i,j) \\ I_x^n(i,j) \end{pmatrix} \qquad \left\| \overrightarrow{N^n}(i,j) \right\| = \sqrt{I_x^n(i,j)^2 + I_y^n(i,j)^2} \tag{4.16}$$

$$\beta^n(i,j) = \overrightarrow{\delta L^n} \cdot \frac{\overrightarrow{N}(i,j,n)}{\left\| \overrightarrow{N}(i,j,n) \right\|} \tag{4.17}$$

y

$$\|\nabla I^n(i,j)\| =$$
$$= \begin{cases} \sqrt{I^n_{xbm}(i,j)^2 + I^n_{xfM}(i,j)^2 + I^n_{ybm}(i,j)^2 + I^n_{yfM}(i,j)^2}, \\ \beta^n(i,j) > 0 \\ \sqrt{I^n_{xbM}(i,j)^2 + I^n_{xfm}(i,j)^2 + I^n_{ybM}(i,j)^2 + I^n_{yfm}(i,j)^2}, \\ \beta^n(i,j) < 0 \end{cases} \tag{4.18}$$

El primer paso dentro del algoritmo será realizar el cálculo de la estimación de la suavidad en dos dimensiones L mediante la ecuación (4.15) y la dirección de los isófotos $\overrightarrow{N}$ mediante (4.16).
Entonces, mediante la ecuación (4.17), se calculará β^n, la proyección de $\overrightarrow{\delta L}$ en el vector $\overrightarrow{N}$.

En el último paso, se multiplicará β^n por una versión de *pendiente limitada* de la norma del gradiente de I, la cual se expresa en (4.18). La motivación de que se utilicen estas *pendientes limitadas* es que la realización de diferencias centrales podría causar una inestabilidad del sistema y, por tanto, estropear la restauración de la imagen.

En el cálculo de estas *pendientes limitadas* todas las componentes poseen un subíndice compuesto por tres letras. Cada una de ellas tiene un significado distinto. La primera de ellas indica la dirección de diferenciación (x o y). La segunda, el sentido de la derivación, siendo b el acrónimo para backward (hacia atrás) y f el de forward (hacia adelante). Y, por último, la tercera letra indica la elección entre el máximo o el mínimo. Este valor será obtenido mediante el cómputo (del máximo o del mínimo) entre el valor y cero.

Finalmente, cabe destacar que la elección de un campo $\overrightarrow{N}$ no normalizado permite un esquema más sencillo y estable. Mediante pruebas con imágenes sintéticas se ha observado que es preferible esta opción ya que existen determinadas situaciones en las que la imagen no llega a reconstruirse de forma adecuada, esto se mostrará más adelante detro del capítulo de "Experimentos y resultados". También cabe destacar que utilizar la versión no normalizada del algoritmo aporta en muchos casos mayor velocidad al proceso y la convergencia se consigue en un número menor de iteraciones.

Ahora bien, cuando el algoritmo de inpainting llega a un estado estacionario $I^n_t = 0$ tendrá el significado de que la *suavidad* es constante a lo largo de los isófotos y la imagen estará completamente reconstruida.

Cuando se aplican las ecuaciones (4.12) a (4.18) a los píxeles en el borde $\partial\Omega$, se conocen parte de los píxeles de fuera de Ω y se utiliza la información que atesoran. Esto es que, conceptualmente se calculan las ecuaciones en una región Ω^ϵ, lo cual es una dilatación de la región Ω una cantidad ϵ, aunque sólo se actualicen los píxeles de dentro de Ω. La información de los píxeles adyacentes dentro de la banda que forma $\Omega^\epsilon - \Omega$ es propagada al interior de Ω, lo cual es fundamental para el funcionamiento del algoritmo.

4.2.2. Difusión Anisótropa para *Inpainting*

Para realizar la difusión anisótropa en *inpainting* se va a utilizar la discretización de la ecuación

$$\frac{\partial I}{\partial t}(x,y,t) = g_\epsilon(x,y)\,\kappa(x,y,t)\,|\nabla I(x,y,t)|\,, \forall x,y \in \Omega^\epsilon$$

Esta discretización se realizará siguiendo los principios enunciados en [12] por Álvarez y cía..

En [12] se propone un estudio de las ecuaciones diferenciales parabólicas no lineales del tipo

$$\frac{\partial u}{\partial t} = g(|G * \nabla u|)\,|\nabla u|\,\mathrm{div}\frac{\nabla u}{|\nabla u|}$$
$$u(x, y, 0) = u_0(x, y)$$

donde $u_0(x, y)$ es la imagen inicial en escala de grises para ser procesada, $u(x, y, t)$ es su versión suavizada de la imagen en un tiempo t, G es un *kernel Gaussiano*, $G * \nabla u$ es un estimador local de ∇u y $g(s)$ es una funcion real no creciente que tiende a cero tal como $s \to \infty$.

Esta ecuación está compuesta por dos términos:

- El término $|\nabla u|\,\mathrm{div}\frac{\nabla u}{|\nabla u|} = \Delta u - \frac{\nabla^2 u(\nabla u, \nabla u)}{|\nabla u|^2}$ representa un término de difusión degenerado que realiza la difusión de u en la dirección ortogonal a su gradiente ∇u y no difunde nada en la direccion de ∇u. Dentro de este término se puede observar que está incluido $\nabla^2 u$, que representa el Hessiano de u.
- El otro término $g(|G * \nabla u|)$ es utilizado para el *realce* de los bordes. Este término controla la velocidad de la difusión al igual que ocurre en [1].

Mediante el uso de modelos anisótropos cuasi-lineales para la detección de bordes llega a una forma cuasi-divergente la ecuación puede ser reescrita

$$\frac{\partial u}{\partial t} = |\nabla u|\,\mathrm{div}\frac{\nabla u}{|\nabla u|}$$

Entonces, con una formulación más literal se obtiene

$$\frac{\partial u}{\partial t} = \frac{\left(u_y^2 u_{xx} - 2u_x u_y u_{xy} + u_x^2 u_{yy}\right)}{||\nabla u||^2}$$

donde los elementos u_x, u_y, u_{xx}, u_{xy} y u_{yy} son derivadas direccionales discretizadas de la siguiente forma

$$u_x(i, j) = u(i + 1, j) - u(i - 1, j)$$

$$u_y(i, j) = u(i, j + 1) - u(i, j - 1)$$

$$||\nabla u||^2 = u_x^2 + u_y^2$$

$$u_{xx}(i, j) = u(i + 1, j) - 2 \cdot u(i, j) + u(i - 1, j)$$

$$u_{xy}(i, j) = \frac{u(i + 1, j + 1) - u(i + 1, j - 1) - u(i - 1, j + 1) + u(i - 1, j - 1)}{4}$$

$$u_{yy}(i, j) = u(i, j + 1) - 2 \cdot u(i, j) + u(i, j - 1)$$

La discretación de esta ecuación nos lleva a

$$\frac{u^{k+1}-u^k}{\Delta t} = \frac{\left(u_y^2 u_{xx} - 2u_x u_y u_{xy} + u_x^2 u_{yy}\right)}{||\nabla u||^2}$$

que, escrito en su forma iterativa resulta

$$u^{k+1} = u^k + \Delta t \frac{\left(u_y^2 u_{xx} - 2u_x u_y u_{xy} + u_x^2 u_{yy}\right)}{||\nabla u||^2}$$

4.2.3. Detalles de implementación

Como ya se ha mencionado anteriormente, este método está compuesto por una aproximación inicial. Tras ello comenzará el método de refinado donde se aplicarán varias iteraciones de *inpainting* además de aplicarse tras éstas un proceso de difusión anisótropa con el objetivo de que la inclusión de valores en el interior de Ω sea suave.
Dentro del bucle de restauración se realizarán A pasos de *inpainting* mediante la ecuación (4.12) y después de ello B iteraciones de difusión. El número total de pasos será T. Es recomendable preestablecer un T máximo o una condición de parada que se active cuando el algoritmo los valores de la actualización de la imagen estén bajo un umbral pre-establecido.

Los valores utilizados en la difusión son $A = 15$, $B = 2$ y $\Delta t = 0{,}1$. El valor de T será dependiente de forma directa del tamaño de Ω.

4.2.3.1. Aproximación multirresolución

En el caso de que Ω sea de un tamaño considerable, Bertalmío en [6] propone utilizar una aproximación multirresolución clásica para acelerar el proceso. En esta aproximación multirresolución se realiza de forma clásica. Se tomará inicialmente una imagen de una resolución más pequeña para arrancar el algoritmo. El resultado del algoritmo en esta imagen de resolución inferior se utilizará para inicializar el algoritmo a una resolución mayor, consiguiendo con ello una mejora en los tiempos de ejecución del algoritmo.

4.2.3.2. Imágenes en color

El tipo de imágenes que se ha estado considerando desde el comienzo de este apartado son las imágenes en escala de grises. Para imágenes en color también se puede realizar el método de inpainting considerando las imágenes en color como conjuntos de tres imágenes. La técnica entonces será aplicada a cada una de las tres imágenes por separado.
Por ejemplo: en caso de que la imagen I estuviese almacenada en formato RGB, el algoritmo se aplicaría a cada una de las distintas capas.

El inconveniente de esta forma de trabajo a RGB es que pueden aparecer colores espúreos cuando el algoritmo ya está bastante avanzado, por ello Bertalmío y cía en [6] proponen utilizar un modelo de color muy similar a LUV puesto que va a utilizar una componente de Luminancia (ρ) y dos componentes cromáticas ($sin\phi$, $sin\psi$). El esquema de transformación de color se muestra en la Figura 4.7.

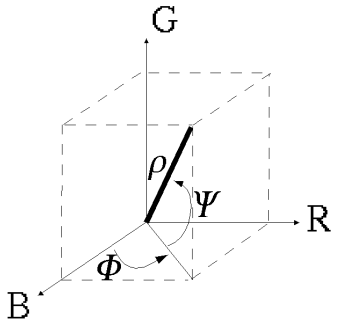


Figura 4.7: Esquema de conversión de color utilizado

La conversión del modelo RGB tradicional a este modelo se puede realizar con las siguientes ecuaciones

$$\begin{aligned} \rho &= \sqrt{R^2+G^2+B^2} \\ sin\,\psi &= \frac{G}{\rho} \\ sin\,\phi &= \frac{R}{\sqrt{B^2+R^2}} \end{aligned}$$

La conversión inversa, para volver a RGB, se realizará con estas otras

$$\begin{aligned} G &= \rho \cdot sin\,\psi \\ R &= \sqrt{\rho^2-G^2}\,sin\,\phi \\ B &= \sqrt{\rho^2-G^2-R^2} \end{aligned}$$

Capítulo 5

Implementación

Según el plan de trabajo expuesto en la introducción de este trabajo, se indicó que para realizar la implementación de los métodos se iba a realizar inicialmente una implementación en un software de análisis numérico y, después de observar un funcionamiento correcto, se procedería a realizar su implementación para un dispositivo móvil.

Partiendo de esa idea, en este apartado se van a dirimir las razones por las cuales se han elegido los entornos y plataformas utilizados en el trabajo.

Además, también se van a incluir unas breves descripciones acerca de las principales herramientas utilizadas durante el desarrollo.

5.1. Análisis numérico

Dentro del campo del análisis numérico existen multitud de software y conjuntos o *suites* que proporcionan herramientas adecuadas para resolver muchos tipos de problemas numéricos. En este caso, lo que se busca es que el software posea propiedades de facilidad en la creación de algoritmos y que también dé facilidades para conseguir manipular imágenes digitales, que es la base principal que se tiene que tratar dentro de los algoritmos.

En este campo existen tres programas que son muy conocidos:

5.1.1. *Scilab*

Scilab es un software matemático que está realizado con un lenguaje de programación de alto nivel. El principal uso para el que esta propuesto es para cálculo científico. Además, se debe añadir que entre sus propiedades que es interactivo, de libre uso y que esta disponible para múltiples sistemas operativos (*Mac OS X*, *GNU/Linux* y *Windows*).

Fue creado para realizar cálculos numéricos aunque también ofrece la posibilidad de hacer algunos cálculos simbólicos como derivadas de funciones polinomiales y racionales. Posee cientos de funciones matemáticas y la posibilidad de integrar programas en los lenguajes más usados (Fortran, Java, C y C++). Fue creado para ser un sistema abierto donde el usuario puede definir nuevos tipos de datos y operaciones entre los mismos.

De las numerosas herramientas que posee cabe destacar: gráficos 2D y 3D, anima-

ción, álgebra lineal, matrices dispersas, polinomios y funciones racionales, programas de resolución de sistemas de ecuaciones diferenciales (explícitas e implícitas), simulador por diagramas en bloque de sistemas dinámicos híbridos, control clásico, robusto, optimización LMI, optimización diferenciable y no diferenciable, tratamiento de señales, grafos y redes, estadística, creación de GUIs, interfaz con el cálculo simbólico (Maple, MuPAD) e interfaz con TCL/TK.

Además se pueden agregar numerosas herramientas o toolboxes hechas por los usuarios.

5.1.2. *MATLAB*

MATLAB es la abreviatura de *MATrix LABoratory*, cuya traducción al lenguaje castellano es "laboratorio de matrices". Se trata de un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas *Mac OS X*, *GNU/Linux* y *Windows*.

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, *Simulink* (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (*toolboxes*); y las de *Simulink* con los paquetes de bloques (*blocksets*).

Para el trabajo que se va a realizar cabe destacar que uno de los *toolboxes* que incluye MATLAB es el de procesado de imágenes que es muy conocido y utilizado en el ámbito.

5.1.3. *GNU Octave*

GNU Octave es programa libre cuya principal función es la de realizar cálculos numéricos. Como indica su nombre es parte de proyecto GNU. MATLAB es considerado su equivalente comercial debido a que comparten varias características y se puede destacar que ambos ofrecen un intérprete permitiendo ejecutar órdenes en modo interactivo. Nótese que Octave usa un lenguaje que está orientado al análisis numérico.

5.1.4. Elección

La elección de *MATLAB* ha sido realizada porque el desarrollador ya se encontraba acostumbrado a él y poseía conocimientos de desarrollo en el entorno. Otro motivo importante por el que *MATLAB* es muy utilizado en el campo del procesado de imagen es debido a la gran cantidad de herramientas que posee el *toolbox* de procesado de imagen que tiene incorporado y también por el *toolbox* de *Wavelets*, que también son muy útiles en el procesado de imagen.

En el caso de que autor hubiese necesitado adecuarse a uno de los tres paquetes de software presentados se hubiese elegido *GNU Octave* debido a las similaridades existentes

con *MATLAB* y las bondades que se presentan sobre él. Otro motivo para elegirlo es la gran cantidad de documentación y ayuda que se puede encontrar en Internet acerca de este software. Como ocurre con la mayoría de proyectos GNU, *Octave* está respaldado por una gran comunidad de usuarios en la que existe una gran interacción y un espíritu de cooperación mediate el cual se mejora lo existente.

5.2. Plataformas móviles

En la actualidad existen varias plataformas *software* para móviles en el mercado:

webOS : Sistema operativo para móviles lanzado por *Palm, Inc.* y posteriormente comprado por *Hewlett-Packard*. Era un sistema multitarea basado en Linux y HTML 5 y desarrollado para dispositivos móviles *Palm*. Tras la compra por parte de *HP*, ésta también lo introdujo en un *tablet*, aunque el dispositivo no tuvo el éxito esperado y desde el 9 de Diciembre de 2011 *HP* renunció a su desarrollo y liberó el código fuente para convertirlo en software libre.

Android : Sistema operativo móvil multitarea basado en Linux que, junto con aplicaciones intercomunicadas entre sí, está enfocado para ser utilizado en dispositivos móviles como teléfonos inteligentes, tabletas y otros dispositivos. Es desarrollado por la Open Handset Alliance, la cual es liderada por *Google*.

BlackBerry OS : Sistema operativo móvil desarrollado por *Research In Motion* para sus dispositivos *BlackBerry*. El sistema permite multitarea y tiene soporte para diferentes métodos de entrada adoptados por *RIM* para su uso en dispositivos móviles, particularmente trackwheel, trackball, touchpad y pantallas táctiles.

iOS : Sistema operativo móvil de *Apple*. Originalmente desarrollado para el *iPhone*, siendo después usado en dispositivos como el *iPod Touch*, *iPad* y *Apple TV*.

Windows Phone 7 : Sistema operativo móvil desarrollado por *Microsoft*, como sucesor de la plataforma Windows Mobile. Está pensado para el mercado de consumo generalista en lugar del mercado empresarial. En otoño de 2012 estará disponible la nueva version 8 solo para nuevos dispositivos.

Del elenco nombrado anteriormente únicamente destacan dos y lo hacen muy por encima del resto: *Android* y *iOS*.
Como se puede observar en la Figura 5.1 entre ambas plataformas la cuota de mercado es mayor al 80 % en Marzo de 2012 y este es el principal motivo para rechazar el resto de ellas. Además, el diferencial que se observa entre Diciembre de 2011 y Marzo de 2012 indica que esta diferencia todavía continúa aumentando y, por tanto, su posición puede llegar a ser todavía más dominante.

Por este motivo se va a realizar un pequeño análisis únicamente de estas dos plataformas, ya que aparentemente el futuro de las plataformas menos utilizadas es dudoso y *Android* e *iOS* están muchísimo más extendidas.

Top Smartphone Platforms 3 Month Avg. Ending Mar. 2012 vs. 3 Month Avg. Ending Dec. 2011 Total U.S. Smartphone Subscribers Ages 13+ Source: comScore MobiLens			
	Share (%) of Smartphone Subscribers		
	Dec-11	Mar-12	Point Change
Total Smartphone Subscribers	*100.0%*	*100.0%*	*N/A*
Google	47.3%	51.0%	3.7
Apple	29.6%	30.7%	1.1
RIM	16.0%	12.3%	-3.7
Microsoft	4.7%	3.9%	-0.8
Symbian	1.4%	1.4%	0.0

Figura 5.1: Estadísticas publicadas por *comScore MobiLens* en su página web correspondientes a las cuotas de mercado en Estados Unidos de las distintas plataformas móviles.

5.2.1. iOS

Esta plataforma está muy extendida mundialmente. Su funcionamiento está basado en Unix y tiene arquitectura multicapa. Esta arquitectura está divida en cuatro capas. De superior a inferior su orden es Cocoa Touch (Capa táctil), Multimedia, Servicios del núcleo y, finalmente, el núcleo del sistema operativo.

El sistema está programado en una variación del lenguaje de programación C denominado Objective-C, el cual es utilizado por *Apple* en todos sus dispositivos.

Para poder realizar desarrollos se ha encontrado que los requisitos son poseer un Mac dotado de *Mac OS X 10.6* o posterior y tener instalado en su interior el *iOS SDK 5.0* o posterior.
Para descargar el SDK (Kit eStándar del Desarrollador) es necesario incribirse en el *iPhone Dev Center*. El SDK contiene el código, la información y las herramientas necesarias para desarrollar, probar, ejecutar y depurar las aplicaciones.
En el interior de este kit existen dos aplicaciones que son fundamentales: *Xcode*, IDE o conjunto de herramientas que permiten editar, depurar y compilar el código fuente, y *Interface Builder*, permite la creación de interfaces gráficos y su vinculación con *Xcode*.

Para probar las aplicaciones desarrolladas mediante *Xcode* se puede utilizar un emulador llamado *iOS Simulator*.
En el caso de querer probarlas en dispositivos reales se deben realizar una serie de gestiones con *Apple*. Una aplicación desarrollada por el usuario no puede ser instalada en ningún dispositivo móvil de forma inmediata. Es necesario realizar una inscripción en el programa para desarrolladores de *Apple* y pagar 99 dólares para recibir una serie de certificados que hagan que la aplicación se pueda instalar en un dispositivo.

5.2.2. Android

Esta es la plataforma más extendida a lo largo del mundo. El funcionamiento de este sistema operativo está basado en una distribución Linux, sobre la que se ejecuta una máquina virtual Java denominada *Dalvik Virtual Machine*. La arquitectura del sistema es multicapa. Sus capas, de la superior a la inferior, son Capa de Aplicación, de Contexto de la Aplicación, de Bibliotecas y Máquina Virtual Dalvik y, en el nivel más bajo, del núcleo Linux.

Cualquier aplicación que se vaya a ejecutar dentro de este sistema operativo debe ser una aplicación escrita en el lenguaje de programación Java, aunque también puede contener fragmentos de código en C/C++.
La inclusión de código en C/C++ se realiza debido a motivos de rendimiento en la mayoría de los casos debido a que este tipo de código es ejecutado como código nativo de la máquina en vez de código interpretado como es Java, y por lo tanto es más rápido realizando las mismas operaciones.

Para realizar desarrollos, *Google* recomienda el uso del IDE *eclipse*, el cual se puede obtener de forma gratuita. Todas las instrucciones de instalación y configuración del entorno para desarrollar se pueden encontrar en "http://developer.android.com".
Además, en esta página también se incluyen los enlaces de descarga de todo el software necesario. Entre este software, *Google* permite descargar de forma gratuita el paquete de desarollo (SDK) de *Android*. Este paquete de desarrollo es descargable para tres plataformas distintas: *Windows*, *Mac OS X* y *Linux*.

El SDK de Android contiene todo lo necesario para poder desarrollar y depurar una aplicación en su ordenador. Dentro del SDK se encuentra un gestor de paquetes mediante el cual se puede descargar el API para cualquiera de las versiones de *Android*.
Para depurar y probar las aplicaciones es necesario un emulador de la máquina virtual *Dalvik*. Este emulador se encuentra incorporado en la SDK de *Android*. Mediante un *plug-in* para *eclipse*, se sintetizan perfectamente la SDK y todas las herramientas que necesita un desarrollador.

En el caso de querer probar la aplicación en un dispositivo real el desarrollador lo único que necesita realizar es crear un fichero de tipo *apk* para la versión del sistema operativo de su dispositivo, copiarlo al dispositivo e instalarlo.

5.2.3. Elección de la plataforma

El autor se ha decantado por utilizar la plataforma *Android* debido a que es más abierta y sencilla frente a *iOS*. El segundo de los motivos para se decantase el autor por esta plataforma ha sido que éste ya poseía conocimentos acerca de programación en Java y algunos conocimientos muy básicos sobre programación en C, frente a los nulos conocimientos que posee sobre Objective-C, que aun siendo un lenguaje de programación similar a C existe el número de diferencias entre ambos para desechar la idea de realizar la adaptación en el poco tiempo disponible. En tercer lugar, se podría establecer como una gran dificultad el desembolso económico que supondría la necesidad de comprar todo el equipo necesario para realizar desarrollos en *iOS*.

5.3. Herramientas para el entorno móvil

Tras haber expuesto los motivos de la elección de *Android* como plataforma móvil en la que se va a desarrollar la aplicación, se va a aproceder a exponer las herramientas utilizadas durante el desarrollo del proyecto.

5.3.1. Android SDK

Son las siglas utilizadas para denominar al Kit eStándar de Desarrollo. Este paquete contiene un gran número de utilidades necesarias poder desarrollar aplicaciones dentro de un sistema.
El gran componente de este paquete es el conjunto de las APIs Java (Interfaz de Programación de Aplicaciones) donde están insertadas gran cantidad de utilidades genéricas que se utilizan a la hora de programar en el lenguaje de programación Java y también se encuentran las funcionalidades genéricas de *Android*.

El segundo componente destacable es un gestor de paquetes de SDK de *Android* mediante el cual se puede realizar la gestión de las APIs que se tienen instaladas en el equipo de desarrollo para cualquiera de las versiones del sistema operativo *Android*.

La última herramienta reseñable que incluye este paquete es el emulador de máquinas virtuales *Android* denominada AVD (de sus siglas en inglés *Android Virtual Device*). Este emulador se ejecuta cada vez que se quieren realizar pruebas de una aplicación *Android* mostrando en una ventana una virtualización de un teléfono móvil de la versión que se haya elegido.
Pero antes de poder ejecutar en cualquier máquina virtual una aplicación desarrollada, se debe crear mediante el gestor *Android Virtual Device Manager* el dispositivo donde se desea probar la aplicación. Este gestor permite crear máquinas virtuales *Android* para cualquiera de las versiones que se han descargado. Una vez creadas se pueden ejecutar modificando ciertos parámetros como puede ser la resolución y también se puede modificar el contenido de la memoria del dispositivo virtual añadiendo (o eliminando) a la máquina virtual ficheros para poder realizar pruebas con él.

Otra herramienta muy útil que se presenta con este conjunto es "adb" (*Android Debug Bridge*), que es una herramienta de línea de comandos que permite al desarrollador comunicarse con una instancia del emulador o conectar con un dispositivo *Android*. Es un programa cliente-servidor que incluye tres componentes: un cliente, un servidor y un demonio.
El cliente se ejecuta dentro del equipo de desarrollo. Se pueden realizar invocaciones a él desde la línea de comandos utilizando el comando "adb".
El servidor se ejecuta como un proceso secundario en el equipo de desarrollo. Este servidor gestiona la comunicación entre el cliente y el demonio "adb" que se está ejecutando en el emulador o dispositivo.
El demonio se ejecuta como un proceso secundario también en cada instancia del emulador o en cada dispositivo.

5.3.2. Android NDK

Este paquete *Android* provee al desarrollador de las conexiones entre el código de la aplicación, que estará escrito en Java, y el código escrito en C/C++. Estas conexiones se desarrollan mediante la API JNI (Java Native Interface), la cual facilita muchas de las operaciones que habría que realizar en el caso de que no existiese. También aporta una serie de librerías C/C++ básicas con las cuales se pueden utilizar dentro de los desarrollos de *Android*.

5.3.3. Eclipse

Eclipse es un IDE (Entorno de Desarrollo Integrado) de código abierto y multiplataforma para desarrollar Aplicaciones. Esta plataforma, típicamente ha sido usada para desarrollar IDEs, como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como *BitTorrent* o *Azureus*.

Es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación. Un ejemplo es el recientemente creado *Eclipse Modeling Project*, cubriendo casi todas las áreas de *Model Driven Engineering*.

Fue desarrollado originalmente por *IBM* como el sucesor de su familia de herramientas para *VisualAge*. Ahora es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Una de las grandes ventajas que presenta Eclipse es la fácil extensión que se puede realizar de sus funcionalidades mediante *plug-ins*. Eclipse posee un amplio y variado catálogo de extensiones mediante las cuales se pueden extender muchas herramientas adicionales. Por ejemplo, para desarrollar aplicaciones *Android* que utilicen código C/C++ existe un *plug-ing* denominado *Sequoyah* el cual ayuda a estructurar el proyecto, ayuda a insertar código en C y también modifica los menús para que el desarrollador tenga mejor acceso a las herramientas necesarias.

5.3.4. Cygwin

Cygwin es una colección de herramientas desarrollada para proporcionar un comportamiento similar a los sistemas Unix en Microsoft Windows. Su objetivo es portar software que ejecuta en sistemas POSIX a Windows con una recompilación a partir de sus fuentes. Aunque los programas portados funcionan en todas las versiones de Windows, su comportamiento es mejor en Windows NT, Windows XP y Windows Server 2003.

Mediante *Cygwin* se pueden ejecutar multitud de comandos Unix. La gran mayoría de sistemas Unix poseen el compilador *gcc*, el cual es muy utilizado por sus usuarios para compilar código fuente en C o C++ y ese es el motivo de uso de esta herramienta dentro de de este proyecto.

Capítulo 6

Experimentos y resultados obtenidos

En este capítulo se pretenden verificar los resultados obtenidos mediante la implementación de los algoritmos propuestos anteriormente.
A parte de verificar el funcionamiento de los algoritmos también se realizará un estudio de la influencia de ciertos parámetros y la toma de ciertas decisiones en la implementación final.

Como muchos otros capítulos dentro de este trabajo, éste se divide en dos partes bien diferenciadas. En la primera de ella se tratarán los resultados presentados por el filtrado de difusión anisótropa y, en la segunda, se presentarán los resultados presentados por el algoritmo de Inpainting.

Dentro de este capítulo se exponen una serie de comparativas realizadas entre los tiempos empleados en la ejecución de los métodos entre los algoritmos diseñados en *MATLAB* un ordenador portátil y los tiempos empleados en la aplicación diseñada para un tablet *Android*. Las especificaciones técnicas tanto del ordenador portátil como del tablet han sido incluídas dentro del Apéndice B.

6.1. Filtrado de difusión anisótropa

A continuación se van a exponer los resultados obtenidos mediante el método semi-implícito propuesto en los capítulos anteriores con el objetivo de realizar una reducción de ruido en la imagen.

En este algoritmo existen tres factores en los cuales el usuario puede actuar para modificar la obtención del resultado. Estos factores son tres: coeficiente de difusividad, paso temporal y el número de iteraciones del algoritmo.
A continuación se expondrá la función y la influencia de cada uno de estos factores en el algoritmo mediante resultados obtenidos. Tras ello se mostrarán más resultados obtenidos mediante este algoritmo y, por último, se expondrá una comparativa realizada entre el tiempo transcurrido para el procesado en el ordenador y en el tablet.

6.1.1. Influencia del coeficiente de difusividad

El coeficiente de difusividad utilizado en el filtrado es tratado como un detector de bordes. Cuanto menor es el valor de este coeficiente, más restrictiva es la detección y más detalles se preservan. Por ende, la difusión tendrá lugar prácticamente en aquellas direcciones perpendiculares a la dirección del gradiente en cada uno de los puntos. Visto de otra forma, la difusión se realizará en las curvas de nivel, remarcando de esta forma los bordes.
En cambio, cuanto mayor es el valor, se pasan por alto un mayor número de bordes y de detalles casi insignificantes. Estos detalles serán eliminados en el caso de que no sean detectados ya que en estas zonas donde no existe un borde la difusión se realizará en ambas direcciones, lo cual hará que se difundan con los valores de su entorno. En los casos de ruido, son tratados como los detalles casi insignificantes que se acaban de nombrar, por tanto, existen muchos casos en los que si no se eliminan los detalles insignificantes, tampoco se elimina el ruido.

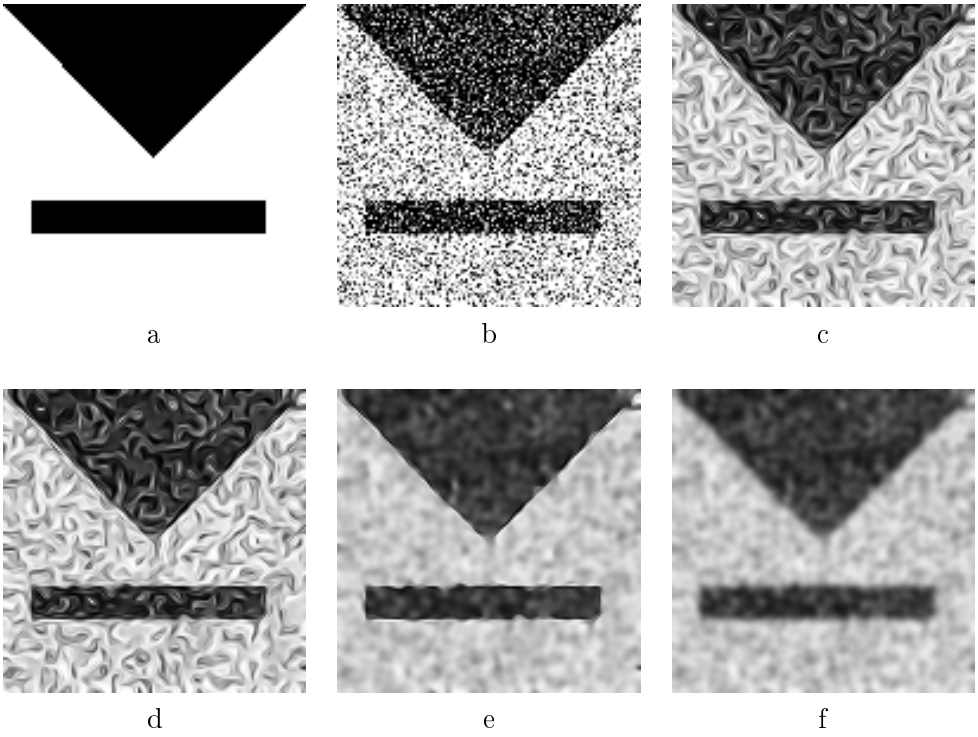


Figura 6.1: (a) Imagen original. (b) Imagen original con ruido gaussiano con $\mu = 0$ y $\sigma = 150$. (c) Restauración $k = 0{,}001$, $\Delta t = 0{,}5$ y iteraciones $= 5$. (d) Restauración $k = 0{,}01$, $\Delta t = 0{,}5$ y iteraciones $= 5$. (e) Restauración $k = 0{,}1$, $\Delta t = 0{,}5$ y iteraciones $= 5$. (f) Restauración $k = 1$, $\Delta t = 0{,}5$ y iteraciones $= 5$.

En la Figura 6.1, se puede observar que este parámetro de detección tiene mucha importancia. En el caso de elegir el parámetro con un valor inferior al necesario, la detección de bordes identifica como si fuesen bordes puntos que en realidad son ruido (como ocurre en las imágenes (c) y (d) de la Figura 6.1), por tanto, es muy importante escoger este parámetro con cierta precisión.
En el caso de escoger el coeficiente muy por encima del valor idóneo para la imagen lo que

ocurrirá es que los bordes se difuminarán, como se puede observar en la imagen (f) de la Figura 6.1.
En el caso (e) de la Figura 6.1, se puede observar que es el parámetro más acertado de todos, puesto que la detección de bordes ha sido mucho más acertada que en el resto de los casos.

6.1.2. Influencia de la elección del paso temporal

Este parámetro indica el avance de la ecuación de difusión en cada iteración. El tiempo final que alcanzará el algoritmo en su evolución temporal vendrá dado por el producto de este parámetro con el número de iteraciones. Como el algoritmo para la resolución de la ecuación utiliza un método incondicionalmente estable, se han probado varios parámetros que no se podrían plantear en el caso explícito.

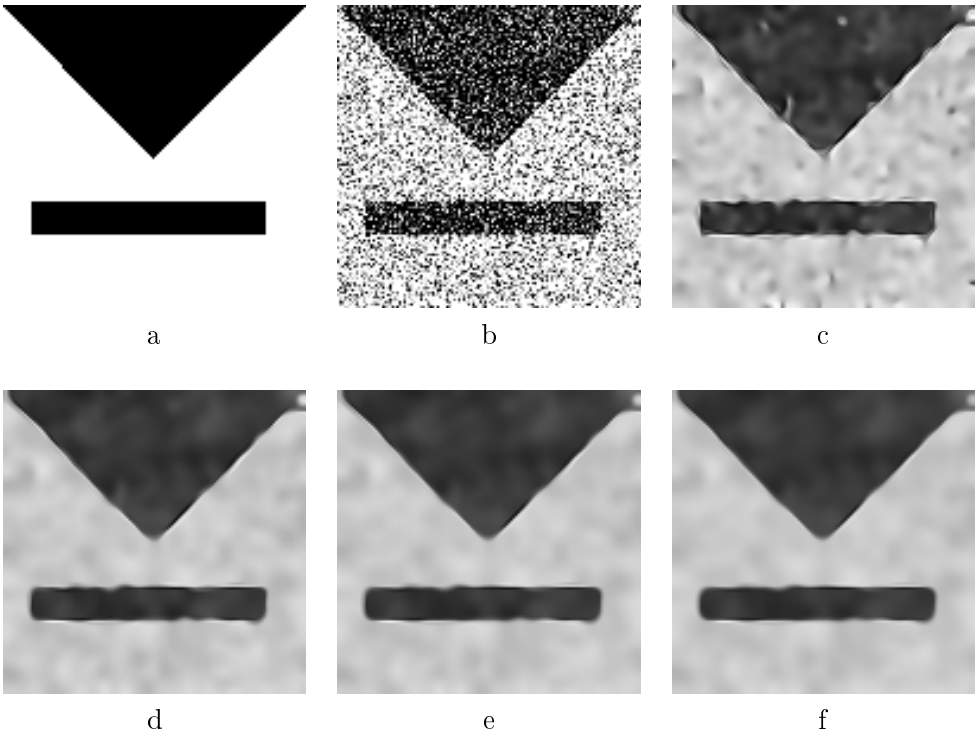


Figura 6.2: (a) Imagen original. (b) Imagen original con ruido gaussiano con $\mu = 0$ y $\sigma = 150$. (c) Restauración $k = 0{,}05$, $\Delta t = 0{,}1$ y iteraciones $= 25$. (d) Restauración $k = 0{,}05$, $\Delta t = 0{,}5$ y iteraciones $= 25$. (e) Restauración $k = 0{,}05$, $\Delta t = 1$ y iteraciones $= 25$. (f) Restauración $k = 0{,}05$, $\Delta t = 3$ y iteraciones $= 25$.

La influencia del parámetro Δt viene dada por que cuanto mayor sea el valor del paso temporal Δt más avanzará la ecuación y el resultado presentará en las áreas planas valores más igualados. También, en los bordes marcados se puede observar que algunos de ellos se suavizan en el caso de tratarse de esquinas con un ángulo bastante acentuado. Por ejemplo, en las imágenes (d), (e) y (f) de la Figura se ve que la esquina inferior del triángulo negro prácticamente se va perdiendo, al igual que ocurre con las esquinas del rectángulo que tienden a curvarse.

En algunos puntos de la imagen pueden surgir artefactos o elementos extraños que, al realizar la difusión, se unifican con estructuras que en realidad sí que existen. Estos artefactos son introducidos por el ruido debido a que en una zona determinada se concentran valores muy similares. Esto ocurre también en la influencia del número de iteraciones con el que se aplica el algoritmo.

Todos estos resultados se pueden observar en la Figura 6.2.

6.1.3. Influencia del número de iteraciones

El número de iteraciones, al igual que el paso temporal Δt tiene una gran influencia en el resultado final. Como se ha contado anteriormente el producto de este con el paso temporal Δt dará como resultado el tiempo final hasta el que evolucionará el algoritmo.

El gran inconveniente de incrementar este parámetro, visto desde la perspectiva de programar para un entorno como *Android*, es el del tiempo que consumirá el algoritmo realizando todo el procesado. Cuantas más iteraciones se den, se consumirá un tiempo mayor.

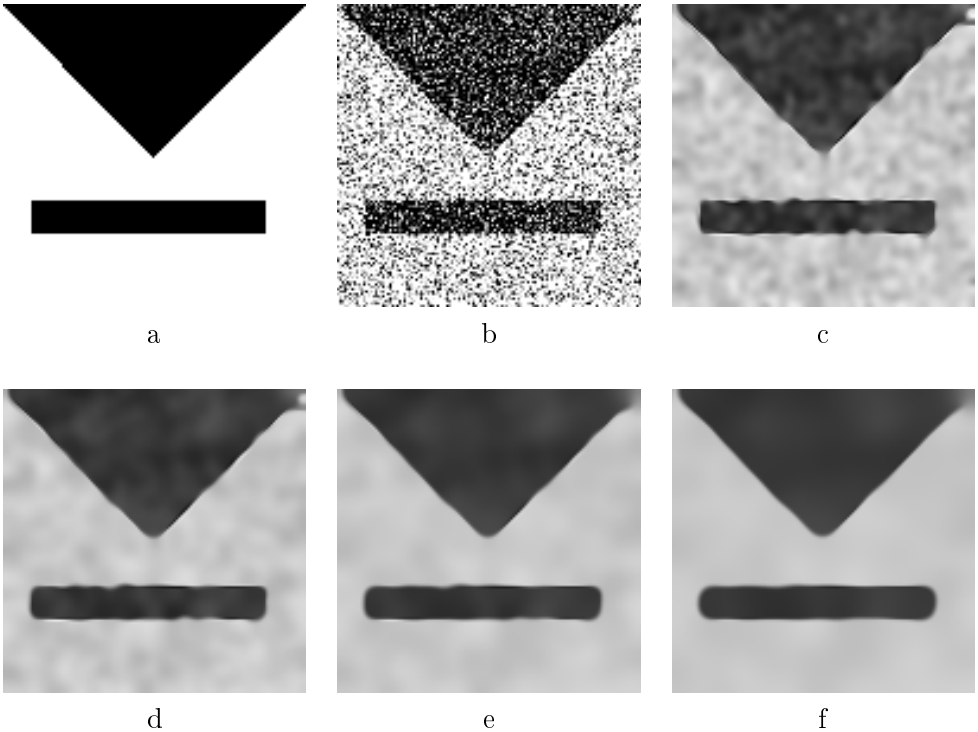


Figura 6.3: (a) Imagen original. (b) Imagen original con ruido gaussiano con $\mu = 0$ y $\sigma = 150$. (c) Restauración $k = 0{,}05$, $\Delta t = 0{,}5$ y iteraciones $= 10$. (d) Restauración $k = 0{,}05$, $\Delta t = 0{,}5$ y iteraciones $= 25$. (e) Restauración $k = 0{,}05$, $\Delta t = 0{,}5$ y iteraciones $= 50$. (f) Restauración $k = 0{,}05$, $\Delta t = 0{,}5$ y iteraciones $= 100$.

Se han realizado varias pruebas y modificando únicamente este parámetro del algoritmo. Los resultados se muestran en la Figura 6.3.

La conclusión principal que se puede extraer es que, para aumentar el tiempo final de

la evolución es mejor incrementar el valor del paso temporal Δt, puesto que con ello no se incrementará la carga del procesador consiguiendo un resultado muy similar.

6.1.4. Resultados con otras imágenes

Todas las pruebas anteriores se han realizado con una imagen sintética exenta de texturas y de regiones donde existiesen bordes poco marcados. En este apartado se muestran los resultados obtenidos utilizando otras imágenes para obtener ejemplos de funcionamiento del algoritmo.

En imagen (a) de la Figura 6.4 se muestra una imagen sintética donde se representa un *phantom*. Esta imagen fue distorsionada introduciendo en ella ruido gaussiano (b), el cual después fue reducido (c) mediante el algoritmo realizado en *MATLAB* con los parámetros que se indican.

En la imagen (c) de la Figura 6.4 se puede observar que los rasgos principales de la imagen han podido ser recuperados, mientras que otros rasgos cuya diferencia tonal con el entorno se funden con el ruido añadido y desaparecen. Esto ocurrirá siempre que se den niveles de ruido elevados.

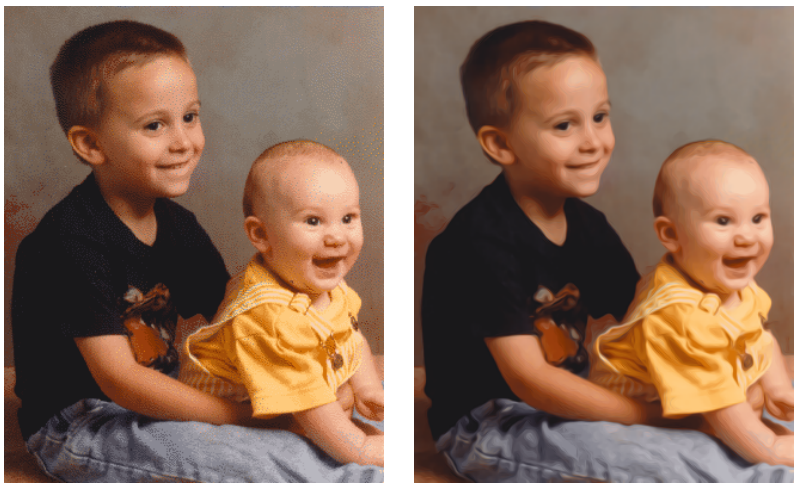
a b c

Figura 6.4: (a) Imagen original. (b) Imagen con ruido gaussiano con $\mu = 0$ y $\sigma = 25$. (c) Restauración $k = 0{,}01$, $\Delta t = 0{,}5$ y iteraciones $= 5$.

En imagen (a) de la Figura 6.5 se muestra la fotografía de dos niños. En ella se puede apreciar claramente una falta de calidad debido a punteados que aprecen en zonas planas. Este punteado seguramente fue debido a que la imagen se almacenó en formato *gif* donde sólo se pueden almacenar 256 colores diferentes.

Para tratar de restaurarla se ha utilizado el algoritmo y se ha obtenido la imagen (b) de la misma Figura. En esta imagen también se puede observar cómo los puntos han desaparecido fundiéndose con los de su entorno, con lo cual el objetivo que se pretendía se ha cumplido. El inconveniente surgido es que se ha producido una gran pérdida de detalles debido a la difusión. Por ejemplo, se han perdido los dientes en el niño mayor, también se han desaparecido costuras y pliegues en la ropa. Otra cosa que también se ha perdido es la textura que anteriormente se podía observar en el pelo de ambos.

En la última de las imágenes utilizadas en los experimentos en la Figura 6.6 (a) se puede observar una gran cantidad de ruido. A pesar de que una cantidad de ruido tan grande es muy difícil de eliminar sin perder muchos detalles, se ha encontrado que se ha



a b

Figura 6.5: (a) Imagen ruidosa. (b) Restauración $k = 0{,}001$, $\Delta t = 0{,}5$ y iteraciones $= 5$.

podido reducir bastante en (b).

Aunque el resultado obtenido no es el deseado, al menos se consigue una mejora bastante sustancial eliminando los granos más pequeños que anteriormente existían en la imagen.

a b

Figura 6.6: (a) Imagen original con ruido gaussiano. (b) Restauración $k = 0{,}005$, $\Delta t = 1$ y iteraciones $= 5$.

6.1.5. Comparativa del coste temporal

Se va a realizar una comparativa entre los métodos realizados en *MATLAB* y ejecutados en el ordenador con los métodos implementados en C para la aplicación *Android* ejecutados en el tablet propuesto anteriormente.
Las comparaciones que se realizarán serán entre las tres últimas Figuras presentadas introduciendo de forma aproximada los mismos valores para los parámetros para obtener resultados visuales lo más similares posible.

Para la Figura 6.4 se ha observado que el algoritmo en *MATLAB* han empleado $t = 0{,}5533$ segundos. En cambio, en el caso del tablet se emplean 2,046 segundos. Se deben tener en cuenta dos factores para razonar el bajo coste de esta imagen. El primero de ellos es que se trata de una imagen pequeña de 128 x 128 píxeles y el segundo es que se trata de una imagen en escala de grises.

Por otro lado, para la Figura 6.5 se ha observado que el algoritmo en *MATLAB* han empleado $t = 1{,}6631$ segundos. En cambio, en el caso del tablet se emplean 17,524 segundos. La motivación del aumento en este caso es que se tiene una imagen a color en formato RGB con una resolución de 318 x 400 píxeles, lo cual aumenta mucho el número de operaciones necesarias con respecto al ejemplo anterior.

Para la Figura 6.6 se ha observado que el algoritmo en *MATLAB* han empleado $t = 3{,}5214$ segundos. En cambio, en el caso del tablet se emplean 32,418 segundos. Este valor también es debido, como en la imagen anterior, a que se trata de una imagen a color con una resolución de 480 x 480 píxeles.

Los tiempos medidos en el tablet son orientativos debido a que se realizan una serie de operaciones que no se contemplan en el algoritmo creado en *MATLAB*. El aumento en el coste temporal siempre viene dado por dos factores muy importantes en las imágenes. El primero de ellos es la resolución de la imagen dada y el segundo presenta grandes diferencias en el caso de que la imagen sea en color o en blanco y negro.

Se puede destacar de este análisis que para imágenes a color el algoritmo en el tablet utiliza un intervalo temporal equivalente a unas diez veces el que utiliza el algoritmo en *MATLAB*.

6.2. *Inpainting*

En este apartado se exponen los experimentos, resultados y ciertas motivaciones en el cálculo de ciertos parámetros.

En el primer lugar se va a proceder a razonar la elección del cálculo del vector $\overrightarrow{N}$. Como se ha podido observar anteriormente, la elección de este vector es crucial para una buena reconstrucción de la imagen.
En segundo lugar se mostrarán los resultados del uso de la aproximación de la solución propuesta y las diferencias que existen entre el algoritmo de Bertalmío [6] y la modificación propuesta en este trabajo.
Por último se expondrán algunos resultados obtenidos mediante los cuales se ilustrará las diferencias de rendimiento entre los algoritmos de *MATLAB* y la aplicación *Android*.

6.2.1. Elección de la dirección de propagación de los isófotos

Anteriormente se expuso que para la elección del vector $\overrightarrow{N}$ existían varias opciones. La primera de las opciones era la elección del vector $\overrightarrow{N}$ como la dirección perpendicular al gradiente normalizada. La segunda, suponía no normalizar el cálculo obteniendo el vector únicamente como un giro de $\pi/2$ radianes del gradiente en el punto y según citaba Bertalmío en [6] era mejor elección que la primera debido a que proporcionaba mejores resultados y un algoritmo más estable. Y, la tercera, propuesta consiste en aplicar un Tensor de

Estructura Modificado mediante el cual se calculará el vector. En esta última propuesta el vector también será normalizado.

Un ejemplo del resultado presentado por las diferentes elecciones se puede observar en las Figuras 6.7 y 6.8.

a b c d

Figura 6.7: (a) Imagen original cuya máscara Ω es la región grisácea. (b) Restauración utilizando $\overrightarrow{N} = \nabla^{\perp} I / \left\| \nabla^{\perp} I \right\|$ (12,93 segundos). (c) Restauración utilizando $\overrightarrow{N} = \nabla^{\perp} I$ (11,4 segundos). (d) Restauración mediante el Tensor de Estructura Modificado (71,9667 segundos).

En la Figura 6.7 la imagen de la que parte la restauración es una imagen en escala de grises con un tamaño de 90 x 96 píxeles de los cuales son seleccionados como máscara 680. Se puede observar que únicamente la imagen (c) debido a la elección del vector $\overrightarrow{N}$ sin normalización consigue restaurar la imagen de forma visualmente adecuada.

En la Figura 6.8 la imagen inicial sobre la que se procesa está en un formato de escala de grises y tiene un tamaño de 405 x 483 píxeles. La máscara que se presenta sobre esta imagen comprende 14258 píxeles. Todo esto hace que el coste computacional pueda incrementarse bastante respecto a la imagen anterior, cuyo tamaño era mucho menor.

Los resultados que presentan todas las imágenes son muy similares.

a b c d

Figura 6.8: (a) Imagen original cuya máscara Ω fue seleccionada para tratar de eliminar todos los defectos producidos por arrugas y rayaduras. (b) Restauración utilizando $\overrightarrow{N} = \nabla^{\perp} I / \left\| \nabla^{\perp} I \right\|$ (641,5569 segundos). (c) Restauración utilizando $\overrightarrow{N} = \nabla^{\perp} I$ (587,91 segundos). (d) Restauración mediante el Tensor de Estructura Modificado (910,25 segundos).

De las imágenes restauradas de las Figura 6.7 y 6.8 también se han medido los tiempos de procesamiento en *MATLAB*. Estos tiempos se muestran una clara reducción del coste temporal dependiente de la elección de $\overrightarrow{N}$. El menor coste se alcanza claramente utilizando la elección más simple del vector. Esto es normal debido a que entraña un número de

operaciones mucho menor.
Con los tiempos de ejecución obtenidos la utilización de un Tensor de Estructura Modificado queda prácticamente descartada para realizar la programación en un dispositivo móvil debido a que el sobrecoste que supone, como muestran las imágenes (d) de las dos Figuras (6.7 y 6.8).
Los tiempos que presentan las imágenes (b) y (c) de las figuras son muy similares. En este caso la ventaja la tiene la imagen (c) donde no se presenta normalización del vector $\overrightarrow{N}$. En lo relativo a los costes de computación se podría elegir cualquiera de las dos opciones.

Pero el factor más determinante para la elección de $\overrightarrow{N}$ es el resultado visual en la Figura 6.7, donde únicamente la elección de $\overrightarrow{N} = \nabla^{\perp} I$ genera un resultado visualmente aceptable. En la Figura 6.8 se puede observar que la restauración en las imágenes (b) y (c) son las que obtienen un resultado visualmente más plausible.

6.2.2. Uso de la aproximación rápida

En esta sección se realiza una comparación de la calidad de los resultados de las restauraciones de los algoritmos de Bertamío [6] y el propuesto que incluye una aproximación inicial a la solución.

Seguidamente se presentan una serie de Figuras (6.9, 6.10, 6.11, 6.12 y 6.13). Cada una de estas Figuras está compuesta de tres imágenes: (a) la imagen original, (b) la imagen restaurada por el método de Bertalmío [6] y (c) la restauración utilizando el algoritmo propuesto en los capítulo anteriores.

a b c

Figura 6.9: (a) Imagen original (96 x 90 píxeles, escala de grises), máscara en tono gris (680 píxeles). (b) Restauración mediante el algoritmo original de Bertalmío [6] (11,4 segundos). (c) Restauración utilizando la aproximación y el refinado (2,31 segundos).

Como se puede observar los resultados en casi todos los casos son muy similares, la reconstrucción en las imágenes (b) y (c) de casi todas las Figuras son muy similares.
Dentro de esta similaridad cabe destacar algunos detalles pertenecientes a la Figura 6.11, que es la que más diferencias presenta entre ambas restauraciones.
Por el lado de la imagen (b), cabe descatar que ha sido mucho mejor restaurado el ojo, donde no se aprecia un borrón.
Por el lado de la imagen (c) se puede apreciar una mejor restauración en la zona del pelo y en las zonas del gorro.
La restauración en el resto de las zonas es más o menos similar.

a b c

Figura 6.10: (a) Imagen original (483 x 405 píxeles, escala de grises), la máscara se seleccionó para eliminar las grietas que presenta la imagen (14258 píxeles). (b) Restauración mediante el algoritmo original de Bertalmío [6] (587,91 segundos). (c) Restauración utilizando la aproximación y el refinado (91,61 segundos).

a b c

Figura 6.11: (a) Imagen original (512 x 512 píxeles, escala de grises), máscara compuesta por las formas circulares de color blanco (3811 píxeles). (b) Restauración mediante el algoritmo original de Bertalmío [6] (943,63 segundos). (c) Restauración utilizando la aproximación y el refinado (142,01 segundos).

Si se comparan los tiempos de restauración, entre un algoritmo y otro, se puede observar que la diferencia es muy sustancial. Obteniendo resultados similares el algoritmo propuesto en este trabajo consigue, mediante la aproximación, obtener unos resultados similares en tiempos muchísimo menores y con ello ayudar a la implementación de este tipo de métodos en dispositivos móviles.

6.2.3. Comparativa del algoritmo *MATLAB* con la aplicación *Android*

El resultado obtenido entre ambos algoritmos es muy similar visualmente, aunque en algunos casos difiere un poco debido a que el giro realizado para el cálculo del vector $\overrightarrow{N}$. En *Android* se ha realizado en sentido contrario al que se realiza en *MATLAB* debido a que la representación de imágenes se realiza de forma distinta.
Esto se muestra en las Figuras 6.15, 6.16, 6.17, 6.18 y 6.19 cuyos originales se pueden contemplar en el punto anterior.

Tras realizar la ejecución de ambos algoritmos se han obtenido los resultados mostrados

a b c

Figura 6.12: (a) Imagen original (483 x 297 píxeles, RGB), la máscara son todos los píxeles donde hay letras rojas (20745 píxeles). (b) Restauración mediante el algoritmo original de Bertalmío [6] (918,09 segundos). (c) Restauración utilizando la aproximación y el refinado (88,13 segundos).

a b c

Figura 6.13: (a) Imagen original (628 x 316 píxeles, RGB), la máscara está compuesta por los trazos blancos (9076 píxeles). (b) Restauración mediante el algoritmo original de Bertalmío [6] (1768 segundos). (c) Restauración utilizando la aproximación y el refinado (178,45 segundos).

en la tabla 6.14. Los tiempos expuestos en *Android* son orientativos comparativamente con los obtenidos en *MATLAB* ya que en los tiempos de la aplicación *Android* están incluidas una serie de operaciones previas y posteriores para gestionar de forma correcta los datos de entrada y salida de la función escrita en el lenguaje C.

	Tamaño Imagen	Píxeles en Ω	*MATLAB*	*Android*
Figura 6.15	96 x 90	680	2,31	1,433
Figura 6.16	483 x 405	14258	91,61	62,111
Figura 6.17	512 x 512	3811	142,01	11,782
Figura 6.18	483 x 297	20745	88,13	89,339
Figura 6.19	628 x 316	9076	178,45	39,459

Figura 6.14: Comparativa en los tiempos de ejecución (expresados en segundos) de los algoritmos *MATLAB* y la aplicación *Android*

Como se puede ver en la tabla de la Figura 6.14 en casi todas las ocasiones es más rápida la aplicación *Android* que el algoritmo *MATLAB*. Esto se debe a que el algoritmo creado en *MATLAB* ha sido optimizado para que ejecute la mayoría de las operaciones de forma matricial, que es para lo que está pensado dicho software de análisis numérico. En cambio, en *Android* las operaciones se han optimizado, realizándolas únicamente en aquellos puntos donde fuese necesario.
Son estos los motivos por los que el número de operaciones en la aplicación *Android* son de-

pendientes del número de píxeles pertenecientes a Ω. En cambio, en los algoritmo *MATLAB* el número de operaciones es dependiente del tamaño de la imagen.

a b

Figura 6.15: Restauración de una imagen sintética (a) mediante el algoritmo en *MATLAB* y (b) utilizando aplicación *Android*.

a b

Figura 6.16: Restauración de una imagen antigua estropeada (a) utilizando *MATLAB* y (b) con la aplicación *Android*.

a b

Figura 6.17: Rellenado de los agujeros en la imagen de Lena (a) con *MATLAB* y (b) usando la aplicación *Android*.

a

b

Figura 6.18: Eliminación de las letras de la imagen de Nueva Orleans utilizando (a) *MATLAB* y (b) la aplicación *Android*.

a

b

Figura 6.19: Rellenado de los trazos en el fotograma de una película de los *Monty Python* (a) aplicando el algoritmo *MATLAB* y (b) ejecutándolo sobre la aplicación *Android*.

6.3. Conclusiones

Como ya se menciona en secciones de capítulos anteriores todo el código desarrollado para esta aplicación ha sido realizado tanto en *MATLAB* para comprender el funcionamiento de los algoritmos y tratar de desarrollar dentro del método ajustes o mejoras tanto para el filtrado de difusión anisótropa como para el método de *inpainting*.
Las imágenes digitales son matrices, bidimensionales o tridimensionales dependiendo del caso, y una de las grandes características de *MATLAB* es la gestión operaciones matriciales. También cabe destacar que el código que desarrolla en *MATLAB* es interpretado y no compilado, por lo que ello repercute en el rendimiento de éste dentro de cualquier equipo o dispositivo.

Bajo estas premisas, el código desarrollado en *MATLAB* ha sido realizado en cualquier lugar posible matricialmente ya que de esta forma sería lo más veloz posible.

Por otra parte, todo el desarrollo de código realizado dentro de la aplicación *Android*, en el lenguaje de programación C, se ha optimizado para obtener una aplicación que utilice el procesador del dispositivo móvil de la forma más eficiente posible evitando de esa forma todas las operaciones innecesarias en puntos en los que no se debe realizar procesado.

En el filtrado de difusión anisótropa, la comparativa entre el código de *MATLAB* y la

aplicación *Android* es muy desfavorable para esta última ya que la optimización expuesta para la aplicación *Android* no tiene aplicación debido a que la difusión se realiza en todos los puntos de la imagen en alguno de los sentidos y se deben recorrer por este hecho todos los puntos de la imagen el número de iteraciones que establezca el usuario.
El coste computacional en *MATLAB*, ejecutado en el ordenador portátil y utilizando cinco iteraciones, se cifra generalmente en unos pocos segundos del orden de dos a cinco debido a la gran cantidad de operaciones matriciales que se pueden implementar en el código. Por contra, en el dispositivo móvil se ha estimado, sin realizar mediciones fidedignas, que es tres o cuatro órdenes superior debido a que, aunque las operaciones han sido optimizadas, el algoritmo requiere de un gran núero de ellas y el procesador tiene también una frecuencia bastante inferior (entorno a tres veces el orden del procesador del portátil).

En cambio, para el método de *inpainting*, en la aplicación *Android* sí que se ha podido aplicar de una forma eficiente una optimización debido a que las modificaciones de los valores de los puntos de la imagen se realizan únicamente en aquellos puntos que pertenecen a la sección a reconstruir, que se ha denominado anteriormente Ω.
En el código en *MATLAB* se han realizado las operaciones de forma matricial, lo cual hace que el tiempo de ejecución no dependa de la región a restaurar Ω, sino que dependerá el tamaño de la imagen dado que las operaciones se realizarán en todos sus puntos.

Como se ha mencionado anteriormente en la Tabla 6.14 se muestra una comparativa de los tiempos de ejecución en algunas de las imágenes que se han tomado de muestra para realizar reconstrucciones de imágenes. Se observa que los rendimientos del código pueden mejorarse con *Android* en comparación con *MATLAB* según la filosofía de programación que se ha utilizado en cada lenguaje.

Capítulo 7

Comparativa con otras aplicaciones *Android*

Dentro del ecosistema *Android* existen multitud de aplicaciones. En el apartado de fotografía se pueden hallar aplicaciones de retoque fotográfico en su mayoría. La característica más abundante en estas aplicaciones de retoque fotográfica es el filtrado artístico.
También se puede destarcar como característica que otras opciones de retoque existentes en las aplicaciones se pueden englobar dentro del campo denominado *registrado de imagen*. En este campo se enblogan operaciones como el recorte en las imágenes, el redimensionado, los giros de imágenes y un largo etcétera.
Otra característica, también destacada y muy común entre las aplicaciones, es que estas imágenes tomadas con el dispositivo móvil pueden ser compartir en las distintas redes sociales que existen en este momento, como son *facebook* o *twitter*.

En esta sección se van a exponer el funcionamiento y las principales características de algunas de las aplicaciones más populares y útiles existentes en *Android*. Esta exposición va a recorrer las aplicaciones evaluando las funcionalidades que estas ofrecen. En ningún momento se va a entrar a evaluar el interfaz de la aplicación debido a que esa cuestión debe ser evaluada en otro entorno y no dentro de la evaluación del desarrollo de los algoritmos de restauración de imágenes.

Finalmente, se expondrán unas conclusiones para realizar una síntesis del análisis de las aplicaciones.

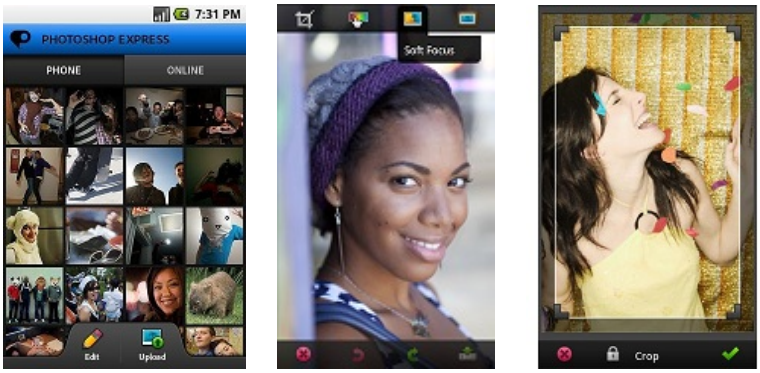
7.1. Aplicaciones de retoque o edición de fotografías

En el ecosistema *Android* existen multitud de aplicaciones relacionadas con el mundo de la fotografía. A continuación se van a exponer las funcionalidades de algunas de estas aplicaciones utilizando como criterio la opinión de los usuarios en el conjunto de las aplicaciones gratuitas que se encuentran en *Google Play*, la tienda de aplicaciones, libros y películas de *Google*.

7.1.1. Photoshop Express

Esta aplicación desarrollada por la empresa *Adobe*, famosa por haber creado el programa de retoque fotográfico más popular para ordenadores (denominado *Photoshop*) en el mundo del diseño gráfico profesional, ha puesto a disposición de los usuarios de forma gratuita un conjunto de herramientas básicas para cualquier amante del retoque fotográfico. Se trata de una hermana pequeña del afamado *Photoshop* para ordenadores.

Las imágenes que permite modificar esta aplicación están englobadas en dos secciones: capturas mediante la cámara y fotografías incluídas en la cuenta de *photoshop.com* del usuario. En ningún caso permite realizar la edición de otras fotografías.



a b c

Figura 7.1: (a) Pantalla de inicio de la aplicación para cargar una fotografía, (b) menú de edición de imagen de la aplicación con acceso a herramientas, y (c) instantánea del recorte de una imagen.

Las herramientas incluidas en la aplicación se agrupan en cuatro grupos: Transformaciones geométricas, Filtros artísticos, Suavizado de la imagen y, por último, Efectos artísticos y de bordes.
En el grupo de las transformaciones geométricas existen las herramientas de Recorte, Giro, Rotación y Simetría.
En el de los Filtros artísticos se puede modificar la Exposición, la Saturación, el Brillo, el Contraste, el Tinte (para cambiar todo el tono de la imagen a un único color) y la transformación de la imagen a Escala de Grises.
En el grupo del Suavizado de la imagen se aplica a la imagen un filtrado paso bajo para suavizar los bordes mediante la herramienta *Soft Focus*.
Para finalizar este apartado de herramientas encontramos el grupo de Efector artísticos y de bordes agrupa modificaciones artísticas de los colores de la imagen así como de los bordes de esta permitiendo varias opciones en cada uno de los apartados.

Por último, cabe destacar, que también existe una aplicación más orientada a profesionales del diseño gráfico denominada *Photoshop Touch*, la cual es más completa en muchos aspectos y permite realizar una edición de fotografía más similar a la edición de *Photoshop* para ordenadores.

7.1.2. Camera 360 Ultimate

Se trata de una aplicación que funciona como una cámara fotográfica digital, llegando incluso a substituir a la aplicación de la cámara nativa que incorpora la distribución *Android* del dispositivo móvil. Permite tomar fotografías de varias formas además de incluir curiosos efectos en ellas. Entre los efectos muchos y variados como pueden ser HDR, corte de color, efectos de película en blanco y negro dentro de un largo etcétera. Este largo etcétera no se limita a los filtros y efectos más conocidos, sino que incluye otros no muy vistos.

También aporta la posibilidad de incluir las fotos tomadas dentro de escenas o paisajes que aporta el programa y que también se pueden descargar de Internet.

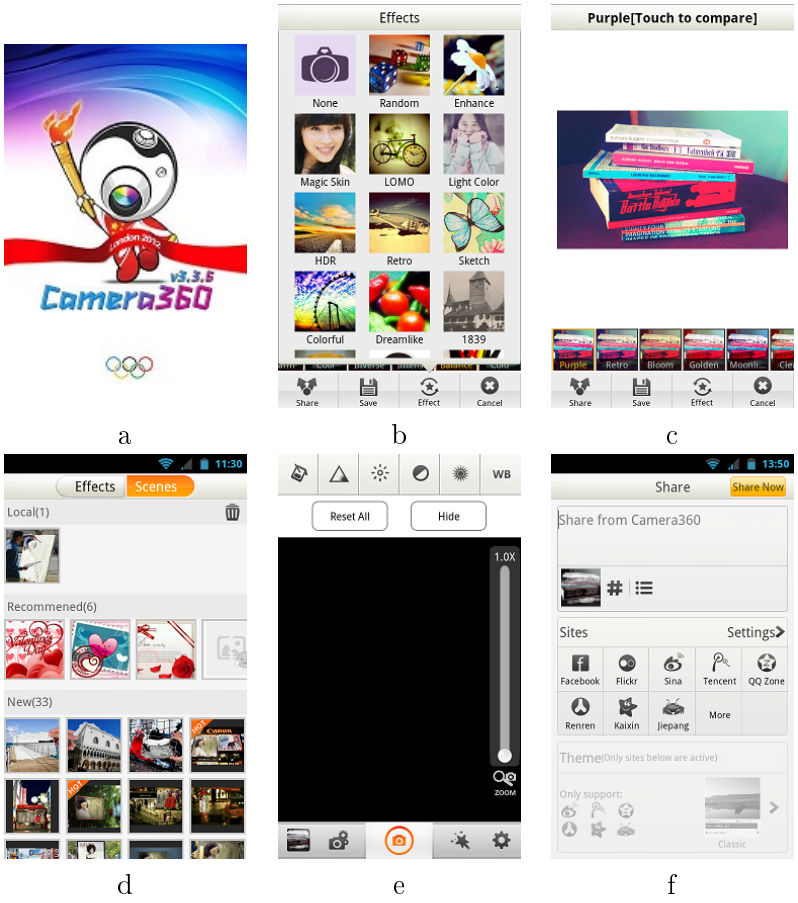


Figura 7.2: (a) Pantalla de carga de la aplicación, (b) malla con los efectos disponibles para aplicar, (c) aplicación de un efecto, (d) escenas predeterminadas para realizar fotomontajes, (e) captura de la opción cámara para tomar una fotografía y (f) opciones para compartir imágenes.

La toma de imágenes se puede realizar de distintas formas y que permite una gestión de cámara no muy habitual en las aplicaciones incluídas en la mayoría de los dispositivos móviles. En este apartado, se pueden observar que están incluídas la toma de imagen en ráfaga, mediante el uso de un temporizador y también de un estabilizador.

Además, el programa también permite cargar imágenes que se encuentran almacenadas

en una carpeta dentro de la memoria del dispositivo cuya ruta es personalizable. Indicando la ruta debida, con esta personalización se puede llegar a retocar cualquier imagen que se encuentre almacenada dentro del dispositivo.

7.1.3. PicsArt - Estudio

Esta aplicación se muestra en la primera posición en el *ranking* de la *Google Play Store*. Es un programa muy bien evaluado por los usuarios ya que tras casi trescientas mil opiniones obtiene una valoración media de 4,7 sobre 5. Esta aplicación aporta una gran cantidad de funcionalidades al usuario.

En la pantalla de inicio permite seleccionar diferentes modalidades. Entre estas modalidades se pueden encontrar las opciones para cargar una imagen, para tomar una fotografía, para insertar efectos, para realizar un collage con varias imágenes y para dibujar sobre una imagen almacenada o sobre un lienzo en blanco. También existen en la aplicación otras funcionalidades para acceder a las redes sociales y compartir en ellas las imágenes.
En el caso de que no se disponga de imágenes en el dispositivo la aplicación también permite el acceso a un repositorio de imágenes propias mediante las cuales hacer los retoques y composiciones deseadas.

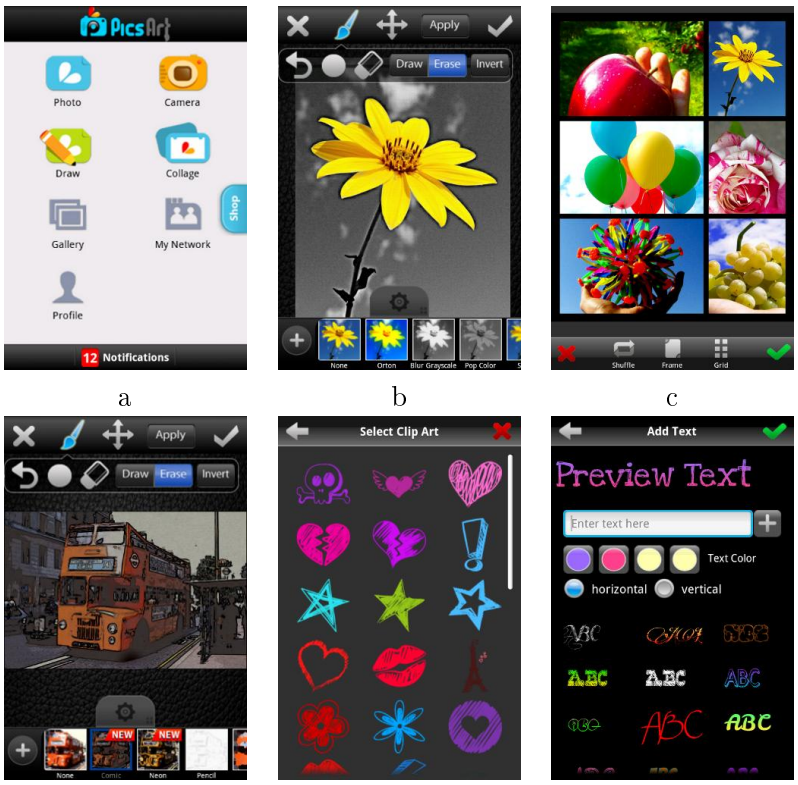


Figura 7.3: (a) Pantalla de inicio de la aplicación, (b) aplicación de un elemento de Efectos, (c) resultado de la creación de una composición, (d) aplicación de una máscara a la imagen de un autobús, (e) selección de ClipArt e (f) inserción de texto.

Dentro de las herramientas para el retoque fotográfico, se pueden observar varios grupos cuando se edita una fotografía: Herramientas de transformación geométrica, Efectos, Máscaras, Bocadillos, Texto, Dibujo, Rótulo, ClipArt, Agregar Foto, Marco y Borde.

Dentro del grupo de las Herramientas de transformación geométrica se pueden encontrar las opciones de Giro, Simetría, Redimensionado y varias opciones de Recorte de las imágenes.
En el segundo de los grupos, los Efectos, se puede encontrar una gran cantidad de efectos artísticos y de corrección agrupados de la siguiente forma: Fx, Artistic, Pop Art, Paper, Color Adjust, Distort, Color Splash y Corrections. Los grupos Fx, Artistic, Pop Art, Distort y Color Splash permiten realizar modificaciones de tipo artístico y dentro de estos se pueden encontrar efectos espectaculares como el HDR, modificaciones para hacer envejecer las imágenes, efectos similares al Pop Art o distorsiones. En los conjuntos restantes, Color Adjust y Corrections, se pueden realizar retoques fotográficos como ajustes de Tono y Saturación, corrección de ojos rojos o eliminación de pequeñas imperfecciones mediante clonación.
El tercero de los grupos define las Máscaras. Estas máscaras son utilizadas para fusionar las imágenes con efectos luminosos entre los que se incluyen formas de neones y círculos, diferentes texturas y también con difeferentes bordes artísticos.
En cuanto a los bocadillos, que están incluidos en el cuarto grupo se pueden añadir dándole a una imagen un aspecto similar el de una viñeta de cómic.
Asimismo, también se pueden añadir textos a la imagen en forma de rótulos u otras indicaciones. Se pueden añadir diferentes fuentes y se pueden personalizar con colores.
En el sexto grupo se encuentra la opción de Dibujo, mediante la cual se pueden realizar dibujos sobre la imagen.
La opción de Rótulo y ClipArt se utilizan para añadir formas predeterminadas que proporciona la aplicación para decorar la imagen.
También existe la opción de Agregar Foto. Esta opción permite crear un mosaico con las imágenes que se encuentra en la memoria del dispositivo móvil.
El penúltimo elememento es Marco, el cual permite añadir diferentes marcos predefinidos en la aplicación a la imagen que se está editando.
Finalmente se encuentra la opción de Borde. Esta opción permite modificar el borde de la imagen añadiendo un borde externo y otro interno. El borde interno se puede definir hacia el exterior de la imagen, para realzar más el borde, y también hacia el interior, en cuyo caso se permite una superposición además de una variación de su opacidad.

El último detalle a destacar de esta aplicación es que permite almacenar las imágenes en la tarjeta SD o en la memoria del dispositivo y también permite compartir las imágenes en diferentes redes sociales muy populares.

7.1.4. Pixlr-o-matic

Esta aplicación publicada por *Autodesk Inc.* es una de las más completas y desarrolladas que se pueden encontrar en el ecosistema *Android*. *Autodesk Inc.* es la empresa responsable de un programa para ordenador muy afamado y utilizado denominado *AutoCAD*. *AutoCAD* es un programa que se utiliza para desarrollar los planos de viviendas y edificios mediantes gráficos vectoriales.

La aplicación publicada en *Google Play* por *Autodesk Inc.* permite al usuario realizar

una serie de fusiones artísticas para personalizar la imagen. Las máscaras que se aplican en estas fusiones están agrupadas en tres conjuntos.
El primero de los conjuntos agrupa las máscaras destinadas a modificar la textura del papel fotográfico de base, modificando con ellos los valores tonales de la imagen.
La fusión de la imagen con diferentes texturas de muy diversa índole, pero sobre todo de iluminación, se encuentran agrupadas en el segundo de los conjuntos.
El tercer y último conjunto permite modificar el marco o borde de las fotografías dándole diversos aspectos artísticos.

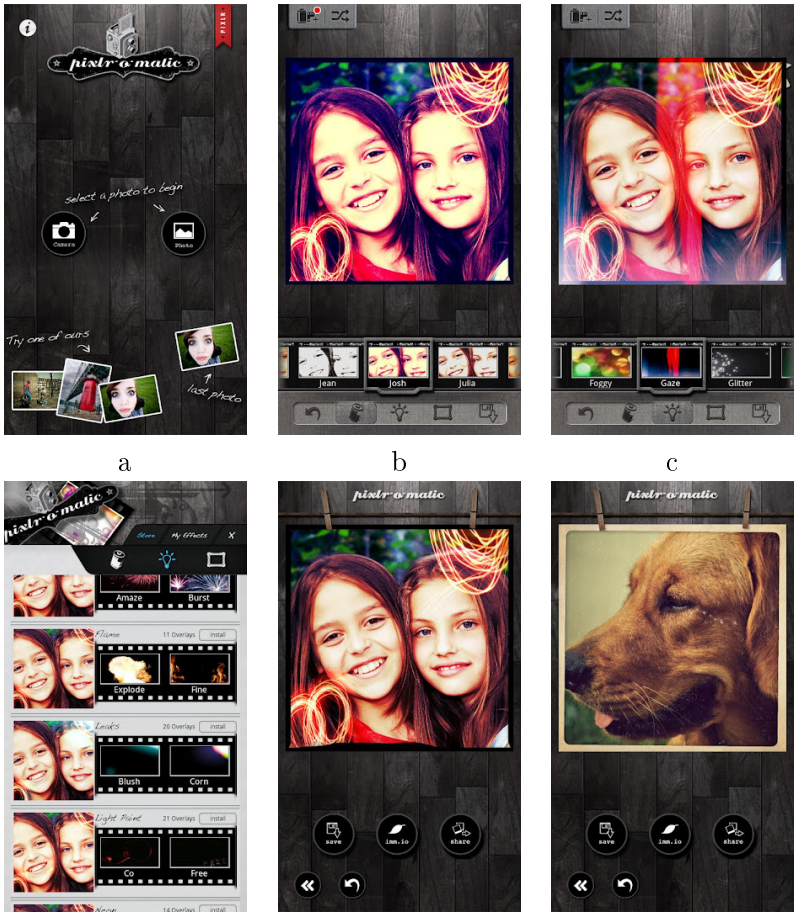


Figura 7.4: (a) Pantalla de inicio de la aplicación donde se selecciona la imagen a cargar, (b) aplicación de máscaras de modo automático, (c) selección de máscara de iluminación en una imagen, (d) gestor de la aplicación para descargar más máscaras para filtrados, (e) almacenamiento/subida a redes sociales de una imagen y (f) de otra imagen.

La aplicación se instala con una serie de máscaras, aunque el número de estos filtros puede ser aumentado descargando conjuntos de máscaras desde la misma aplicación para cualquiera de los tres conjuntos de filtros.

Esta aplicación también permite el almacenamiento de imágenes en el dispositivo. Además, como muchas otras también permite compartir las imágenes mediante otras aplicaciones que se encuentren instaladas en el dispositivo, como son clientes de correo electrónico,

redes sociales y otros.

7.1.5. Photo Editor

Se trata de una potente aplicación localizada en el segundo puesto del *ranking* de la categoría de fotografía dentro de la tienda *Google Play*. Es una aplicación sencilla que integra las herramientas más comunes en el retoque fotográfico.

En la pantalla inicial muestra varias opciones para cargar imágenes y poder editarlas: carga desde la galería de imágenes, desde la cámara del dispositivo y desde un explorador de archivos. También muestra una opción mediante la cual se puede crear una nueva imagen para realizar en ella cualquier diseño que se desee.

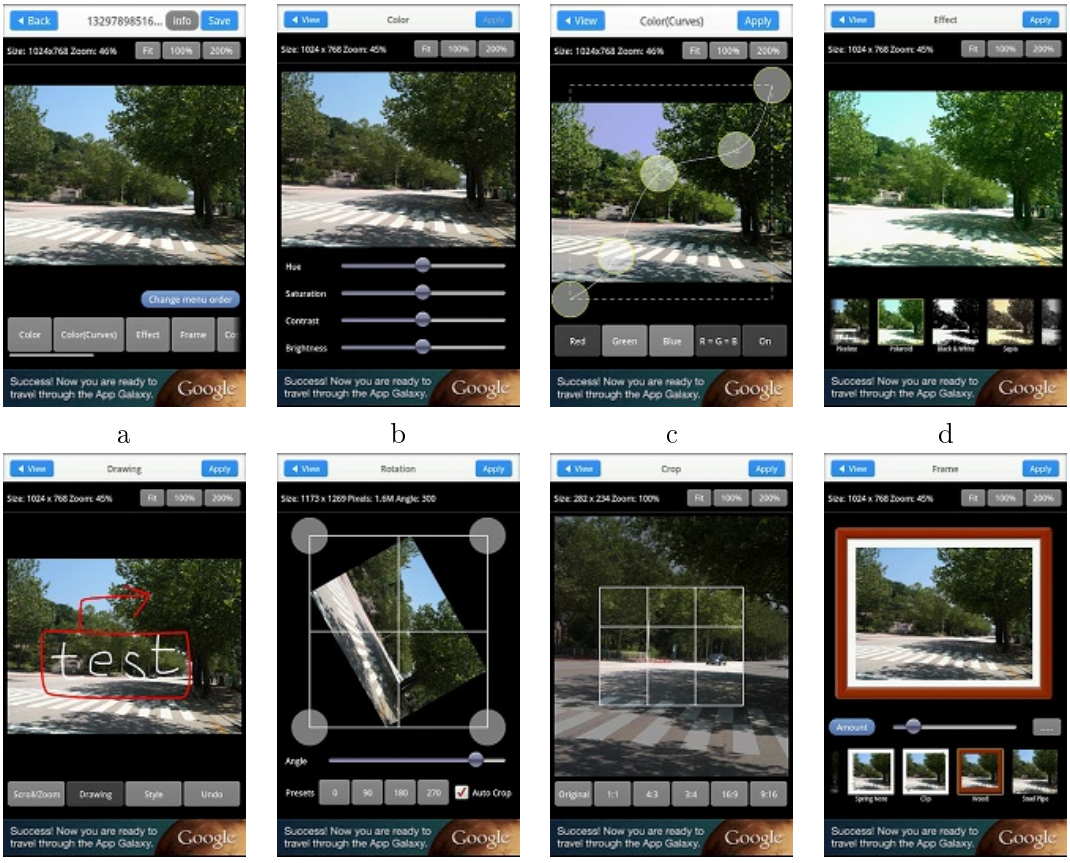


Figura 7.5: (a) Menú de edición de la imagen, (b) modificación de los colores, (c) modificación de la Curva de colores, (d) aplicación de un Efecto, (e) ejemplo de Dibujo, (f) de Rotación, (g) de Recorte y (h) edición de los Marcos.

Una vez cargada la imagen el programa aporta una serie de opciones para la edición que agrupa en Color, Curvas de Color, Efectos, Marcos, Correcciones, Dibujo, Texto/Imagen, Rotación, Cortar y Escalar. Como se ha enunciado anteriormente, prácticamente todas las opciones que muestra el programa representan las básicas en una aplicación de retoque fotográfico, las cuales se basan en técnicas sencillas de procesado de imagen y de

transformaciones geométricas.
El primero de los grupos se denomina Color y en su interior se pueden realizar modificaciones básicas de la imagen. Las modificaciones que se pueden son las de Tono, Saturación, Brillo y Contraste.
La opción de Curvas de Color permite al usuario realizar un ajuste manual de las curvas de color para cada una de las componentes por separado.
En tercer lugar se encuentra la opción de Efectos, donde se incluyen efectos como modificación del Gamma de la imagen, Contraste y Tono automáticos, Desenfoque, Enfoque, Borde, Blanco y Negro, Inversión y una serie de filtros artísticos.
Los Marcos, como se puede intuir por su nombre, añaden marcos a la imagen de diferente tipo. Existen una serie de marcos predeterminados en la aplicación. Dentro de las opciones existentes se pueden realizar pequeñas modificaciones.
El conjunto que se presenta en el número cinco de la lista es el de Correcciones. Las correcciones que se permiten en este apartado son la de la Temperatura del color de la imagen y también la corrección de ojos rojos.
El siguiente elemento en la lista es el Dibujo. Mediante este elemento se pueden dibujar trazos sobre el lienzo pudiendo elegir el grosor y la opacidad del trazo.
La aplicación también permite añadir texto e imágenes a la composición para realizar un *collage* o composición mediante la opción del menú Texto/Imagen.
Las últimas opciones de la lista son transformaciones geométricas de la imagen como la Rotación, el recorte y el Escalado de la imagen. La primera de esta terna permite girar la imagen respecto de su punto central. El recorte permite eliminar zonas de la imagen. Y, por último, la opción de Escalado permite modificar el tamaño de la imagen en número de píxeles y, por tanto, el tamaño que después ocupe ésta en la memoria del dispositivo.

En cuanto al almacenamiento, la aplicación permite un buen número de formas diferentes. Se pueden seleccionar dos formatos de almacenamiento distinto para las imágenes. Uno de los formatos tiene pérdidas de calidad, el cual es JPEG.El otro formato es PNG, el cual no soporta pérdidas.
El formato JPEG permite almacenar las imágenes en la Galería del dispositivo o en la tarjeta SD. También permite establecer la imagen creada como fondo de escritorio del sistema *Android*. La última opción que presenta es la de Compartir. Con esta opción se puede utilizar la imagen en cualquiera de las aplicaciones instaladas en el dispositivo que sean compatible con el formato.
Por su parte el formato PNG permite únicamente Compartir las imágenes y almacenarlas en la tarjeta SD.

7.1.6. Otras aplicaciones

Dentro del listado de aplicaciones para *Android* dentro del conjunto dedicado a fotografía, donde se encuentran las aplicaciones relacionadas con el procesado de imagen, se han querido destacar las anteriormente expuestas, aunque existen otras muchas e incluso se podrían encontrar otras herramientas a destacar de ellas.

Dentro de esas herramientas se podrían poner como ejemplo las aplicaciones *Pho.to Lab* o *PhotoFunia*. Estas dos aplicaciones emplean algoritmos de segmentación para realizar la detección de rostros para realizar composiciones fotográficas añadiendo el rostro detectado en la imagen a una plantilla predeterminada incluida en la aplicación.

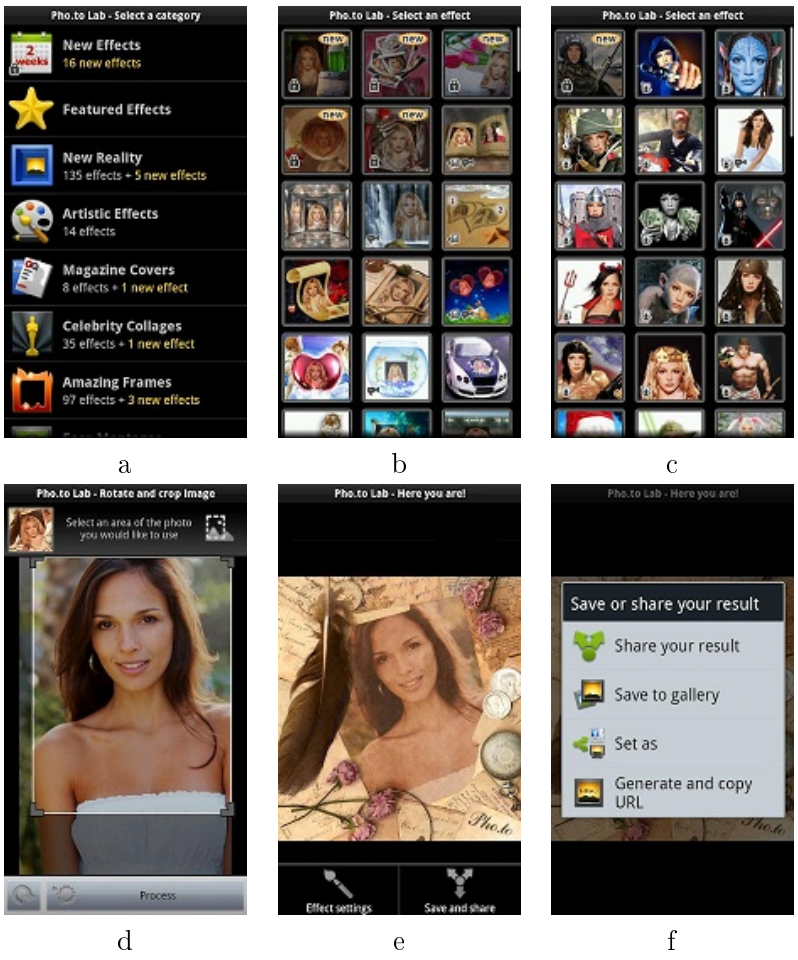


Figura 7.6: **Photo.Lab** (a) Menú inicial de la aplicación para selección de conjunto de efectos, (b) selección de efectos varios, (c) selección de efectos con reconocimiento facial, (d) selección de la región de la imagen que se desea fusionar, (e) resultado final ya procesado y (f) almacenamiento de la imagen resultado.

Otra herramienta bastante apreciada por los aficionados a la fotografía es el HDR (*High Dynamic Range* o amplio rango dinámico). Esta herremienta se utiliza mejorar el rango de las luminancias entre las zonas más claras y más oscuras de la imagen y conseguir con ello realzar detalles ocultos en la imagen. Esta técnica también es utilizada para realizar efectos artísticos en las imágenes. Un ejemplo de la aplicación de esta técnica se da en la aplicación *HDR Camera*.

7.2. Conclusiones

En esta sección se evaluará el estado general de las aplicaciones anteriormente expuestas y se realizará una comparación con las herramientas desarrolladas en la aplicación *PhotoRestore*, la cual es el fruto final de esta tesina.

Las herramientas que se han expuesto realizan todas procesado de imagen de una forma u otra. Algunas realizan filtrados de la imagen mediante diferentes máscaras de convolución. En este grupo se puede englobar los filtrados de suavizado, como también los de emborronado de imágenes o los de detección de bordes.
En el campo del registrado de imagen también se realizan transformaciones geométricas sencillas. Ésas se incluyen en los grupos de las transformaciones rígidas, afines y de escalado.
Como se ha podido observar, uno de los cometidos más destacados de todas las aplicaciones es realizar filtrados artísticos mediante fusiones de la imagen con máscaras de muy diverso tipo. En general, estas máscaras modifican los tonos de la imagen añadiendo nuevas texturas a la imagen y fusionándose con los ya existentes, consiguiendo con ellos efectos como el envejecimiento de la imagen. Otra característica es que modifican los marcos de las imágenes añadiendo toques artísticos a éstas cuyo resultado es estéticamente muy vistoso.

En ningún momento se ha encontrado ninguna aplicación que posea herramientas para realizar restauraciones de imágenes mediante cualquier procedimiento relacionado con el procesado de imagen.

La aplicación creada, *PhotoRestore*, se centra en dos herramientas de procesado avanzado de imágenes digitales: el *filtrado de difusión anisótropa* y una técnica de *inpainting*. Ninguna de estas técnicas había sido implementada todavía en *Android*. Es por este motivo, que las herramientas incluidas en la aplicación creada es una novedad dentro del ecosistema *Android*.

La primera de las herramientas, el *filtrado de difusión anisótropa*, ha sido creada de tal forma que el usuario de la aplicación pueda seleccionar los parámetros básicos para el filtrado: el *coeficiente de difusión*, el *tamaño de paso* de cada iteración y el número de iteraciones que se realizarán. Esto hace que la herramienta sea más versátil de lo normal. El cometido inicial de la herramienta es el de poder reducir el ruido de la imagen seleccionando los parámetros adecuados. Pero además, el usuario también puede crear, a partir de la imagen que desee filtrados artísticos tendiendo a dos opciones: crear imágenes únicamente con regiones planas realizando una difusión de gran magnitud un gran número de iteraciones utilizando un coeficiente de difusividad muy alto o hacer un filtrado de la imagen donde se detecte muy bien cualquier variación que marque un borde en la imagen, incluso el ruido será detectado como un borde, y se difunda de forma casi única en la dirección paralela a los bordes. Esta última opción puede dar lugar a creaciones artísticas que se asemejarán a los cuadros creados por Vincent Van Gogh, como el cielo del cuadro "Noche estrellada".

Por su parte la herramienta de *inpainting* ha sido implementada para que su ejecución sea más rápida que el planteamiento inicial realizada por Bertamío, mediante una aproximación rápida de la solución como paso inicial. Gracias a esto, se consigue que los tiempos de ejecución del algoritmo dentro de un dispositivo con una potencia de cómputo muy limitada se vean muy reducidos presentando unos resultados muy similares a los del algoritmo original.

Además, cabe resaltar que las herramientas creadas han sido optimizadas en el mayor grado posible para hacer que su ejecución sobre un dispositivo con una capacidad bastante limitada de procesamiento sea correcta y con la mayor celeridad permitida.

En comparación con las aplicaciones anteriormente expuestar, se deberían depurar muchas cosas del interfaz gráfico, cosa que se realizará más adelante ya que el autor tiene la intención de mejorar la aplicación añadiendo funcionalidades. Para cualquier usuario de *Android* el interfaz gráfico existente en estos momentos es algo lioso ya que el usuario no dispone del acceso a las herramientas directamente en la pantalla donde se muestra la imagen que se quiere restaurar.
Aunque este trabajo no se centraba en la creación de la aplicación *Android* en sí, sino en las técnicas de restauración de imágenes que funcionan detrás de todo el entramado, también se quiere hacer hincapié en que una aplicación para dispositivos móviles debe ser sencilla de utilizar y también intuitiva, cosa que no cumple la aplicación desarrollada.

También, si se hace uso de las aplicaciones anteriormente indicadas, la mayoría de ellas ejecutan las herramientas de una forma casi instantánea debido a que las operaciones realizadas no deben ser computacionalmente muy costosas. Es por ello que la aplicación creada parte con una clara desventaja con ellas.
Por otra parte, la aplicación *Pho.to Lab* que realiza fusiones de imágenes también tiene que realizar técnicas avanzadas de procesado de imagen para detectar las posiciones de los rostros en las imágenes y después registrarlas de forma automática a una posición determinada en dependencia de la imagen. Todo este procesado hace que los tiempos de ejecución también se disparen para esta aplicación.

Capítulo 8

Conclusiones, aportaciones y líneas futuras

8.1. Conclusiones

Como se ha visto durante todo el desarrollo de esta Tesina, el objetivo principal de ésta era la adaptación de técnicas de procesado avanzado de imagen a los dispositivos móviles. Este objetivo se quería afrontar de forma práctica resolviendo dos problemas relacionados con las fotografías digitales, por un lado la reducción de ruido mediante el *filtrado de difusión anisótropa* y, por el otro, la restauración de regiones o eliminación de elementos indeseados mediante *inpainting*.

8.1.1. Filtrado de difusión anisótropa

Respecto al *filtrado de difusión anisótropa* se realizó el estudio del estado del arte donde se encontró que destacaban dos filtros: el creado por Perona y Malik [1] debido a que abrieron un gran campo de estudio mediante el uso de EDPs y el filtro de Weickert [2] donde mejoró en gran manera la calidad del filtrado gracias a la introducción del Tensor de Difusión para implementar la anisotropía en la imagen y el esquema semi-implícito para asegurar una evolución correcta evolución de la ecuación de difusión.

El método implementado para el filtrado de difusión anisótropa tiene muy en cuenta la anisotropía que presenta la imagen medida por los autovalores del tensor de estructura. También muestra una gran estabilidad gracias al esquema semi-implícito propuesto.

La dificultad principal para obtener buenos resultados en este tipo de algoritmos residirá en la elección que se haga del *coeficiente de difusividad*. Anteriormente se ha mostrado su influencia y este valor se ha dejado a elección del usuario para que indique hasta qué punto quiere preservar los detalles de la imagen.

Este filtro también podría utilizarse como un filtro artístico debido al papel que se puede conseguir que tenga este *coeficiente de difusividad*. Con él se podrían tanto emborronar la imagen, como crear texturas inexistentes previamente.

8.1.2. *Inpainting*

En cuanto a la otra aplicación práctica desarrollada, *inpainting*, se ha podido observar que existen multitud de enfoques para conseguir un resultado aceptable mediante el estudio del estado del arte. Cada uno de estos enfoques provee resultados muy diferentes para entradas similares.

Como se ha observado, la elección del cálculo de los diferentes factores puede dar lugar a resultados poco aceptables para la vista en condiciones extremas. Estas elecciones de los cálculos deben ser realizadas con sumo cuidado y deberían ser testadas en un número mayor de imágenes para comprobar su buen comportamiento.

Por un lado, se debe destacar que la aproximación a la solución que se realiza no obtiene unos resultados muy brillantes, aunque ayuda a que el conjunto del proceso se acelere mucho obteniendo unos resultados muy similares a los del algoritmo original.

Por otro lado, también cabe destacar la mejora en el coste temporal en la aplicación *Android* gracias a la optimización realizada en los cálculos. En *Android* únicamente se realizan operaciones sobre los puntos de la máscara, lo cual hace que no se puedan mejorar mucho más los tiempos de ejecución del algoritmo.

8.1.3. Programación en *Android*

El autor de esta Tesina en ningún momento había tratado la programación para plataformas móviles. Es por ello que desconocía el rendimiento que este tipo de dispositivos podía manifestar cuando se ejecutasen sobre ellos técnicas de procesado de imagen. Además, también se desconocía el formato de gestión de los datos de las imágenes en este tipo de equipos.

La plataforma *Android* utiliza el patrón *Model-View-Controller* para estructurar las aplicaciones, como realizan gran cantidad de *frameworks Java*. De esta forma se consigue realizar una separación clara de los tres módulos de la aplicación: el Modelo, la forma de la cual se almacenan los datos; la Vista, la interfaz gráfica con la que interactúa el usuario; y el Controlador, las operaciones que se deben ejecutar para cada acción que indica el usuario. Esta separación hace muy sencillo poder modificar y extender las aplicaciones ya existentes mediante la inclusión de pocas líneas de código.
La programación relativa a ciertos aspectos de gestión de datos y mensajes está muy madura debido a la aportación de todas las *APIs* de Java y todo el tiempo y esfuerzo que se lleva desarrollando el lenguaje de programación Java.
En cambio, otros aspectos como son el diseño de las interfaces gráficas interactivas y cierto tipo de acciones sobre ellas que parece que podrían estar integradas deben ser programadas desde cero, lo cual complica en parte las cosas.

Aún con todo ello, el hardware de los dispotivos móviles que existen hoy en día todavía puede resultar insuficiente para realizar tareas de procesado avanzado de imágenes con una gran carga en procesamiento. Quizá futuros dispositivos puedan llegar a hacer posible que este tipo de procesados sean muy ágiles mediante algoritmos multihilo para los múltiples núcleos que incorporen sus procesadores.

8.2. Aportaciones

Las aportaciones que se han realizado durante la realización de este trabajo han sido variadas y se muestran en la siguiente lista.

- Estudio de los métodos ideados hasta el presente en los campos del *filtrado de difusión anisótropa* y de *inpainting*.
- Adaptación y creación de una variación del método de *inpainting* de Bertalmío y cía. [6] mediante una aproximación inicial utilizando planteamientos de Telea [10] para acelerar el procesado del algoritmo.
- Implementación de los métodos expuestos en *MATLAB*.
- Diseño e implementación de los métodos de *filtrado de difusión anisótropa* y *inpainting* optimizados en el lenguaje de programación C.
- Demostración de la influencia de los parámetros que puede introducir el usuario en el *filtrado de difusión anisótropa*
- Validación de los resultados para elegir adecuadamente el cálculo de algunos factores determinantes en el método de *inpainting*.

8.3. Líneas futuras

Continuando con las líneas del trabajo de esta Tesina, es necesario mejorar las técnicas propuestas por los algoritmos con el objetivo de obtener mejores resultados, tanto en el coste temporal como en el visual. La aplicación *Android* también podría mejorarse añadiendo nuevas funcionalidades.
En un futuro se pueden dar posibles líneas siguientes de mejora:

- Inclusión de técnicas probabilísticas en el *filtrado de difusión anisótropa* para reducir los tiempos de procesado necesarios.
- Modificación del cálculo de la detección de los bordes de la imagen en el *filtrado de difusión anisótropa* para tratar de mejorar la preservación de detalles de la imagen.
- Modificación de la aproximación a la solución del método de *inpainting* utilizando un método rápido basado en parches para mejorar los resultados cuando las regiones Ω a restaurar son grandes.
- Extensión del algoritmo de *inpainting* para lograr restaurar la textura, además de restaurar el color y los bordes más destacados de la imagen.
- Implementación de otras técnicas de procesado avanzado de imágenes, como pueden ser técnicas de *deblurring*, para realizar un enfoque de imágenes borrosas. El desenfoque de imágenes ocurre muchos casos en el mundo de la fotografía digital.
- Creación de filtros artísticos y filtros orientados al realce de imagen, que parecen estar de moda en la actualidad en las aplicaciones móviles, en conjunción con la posibilidad de poder compartir las fotos en una red social.

Bibliografía

[1] P. Perona y J. Malik. *Scale-space and Edge Detection Using Anisotropic Diffusion.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, num. 7 (Julio 1990).

[2] J. Weickert. *Anisotropic Diffusion in Image Processing.* ECMI Series, Teubner-Verlag, Stuttgart, Germany, (1998).

[3] J. Weickert, B. M. ter Haar Romeny y M. A. Viergever. *Efficient and Reliable Schemes for Nonlinear Diffusion Filtering.* IEEE Transactions on Image Processing, vol. 7, num. 3, (Marzo 1998).

[4] J. Weickert. *Coherence-Enhancing Diffusion of Colour Images.* Image Vision Comput., vol. 17, num. 3-4, páginas 201-212, (1999).

[5] M. Bertalmío. *Processing of flat and non-flat image information on arbitrary mainfolds using Partial Differential Equations.* Universidad de Minessota, (Marzo de 2001).

[6] M. Bertalmío, G. Sapiro, V. Caselles y C. Ballester. *Image Inpainting.* SIGGRAPH (2000).

[7] M. Bertalmío, A. Bertozzi y G. Sapiro. *Navier-Stokes, fluid dynamics, and image and video inpainting.* In Proc. IEEE Int. Conf. on Comp. Vision and Pattern Recog., Hawaï, (2001).

[8] M. Bertalmío, L. Vese, G. Sapiro y S. Osher. *Simultaneous structure and texture image inpainting.* IEEE Transactions on Image Processing, 12(8), páginas 882-889, (2003).

[9] M. M. Oliveira, B. Bowen, R. McKenna y Y.-S. Chang. *Fast Digital Image Inpainting.* Proceedings of the International Conference on Visualization, Imaging and Image Processing, (2001).

[10] A. Telea. *An Image Inpainting Technique Based on the Fast Marching Method.* Journal of Graphics Tools, vol. 9, num. 1, páginas 23-24 (2004).

[11] F. Bornemann y T. März. *Fast image inpainting based on coherence transport.* J. Math. Imaging and Vision, 28(3), páginas 259-278, (2007).

[12] L. Álvarez, P.-L. Lions y J.-M. Morel. *Image selective smoothing and edge detection by nonlinear diffusion.* SIAM J. Numer. Anal. 29, páginas 845-866, (1992).

[13] A. A. Efros y T. K. Leung. *Texture synthesis by non-parametric sampling.* In Proceedings of the International Conference on Computer Vision, volumen 2, página 1033. Citeseer, (1999).

[14] A. Criminisi, P. Pérez y K. Toyama. *Region filling and object removal by exemplar-based inpainting.* IEEE Transactions on Image Processing, 13(9), páginas 1200-1212, (2004).

[15] M. Ashikhmin. *Synthesizing natural textures.* Proc. ACM Symp. Interactive 3D Graphics, ACM Press, páginas 217-226, (2001).

[16] J. Sun, L. Yuan, J. Jia y H.Y. Shum. *Image completion with structure propagation.* In ACM SIGGRAPH 2005 Papers, página 868. ACM, (2005).

[17] C. Barnes, E. Shechtman, A. Finkelstein y D.B. Goldman. *Patchmatch: A randomized correspondence algorithm for structural image editing.* ACM Transactions on Graphics, 28(3), 2, (2009).

[18] M. Elad, J.L. Starck, P. Querre y D.L. Donoho. *Simultaneous cartoon and texture image inpainting using morphological component analysis (MCA).* Applied and Computational Harmonic Analysis, 19(3), páginas 340-358, (2005).

[19] M. Aharon, M. Elad y A. Bruckstein. *K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation.* IEEE Transactions on signal processing, 54(11), página 4311, (2006).

[20] J. Mairal, M. Elad y G. Sapiro. *Sparse representation for color image restoration.* IEEE Transactions on Image Processing, 17(1), página 53, (2008).

[21] J. Mairal, G. Sapiro y M. Elad. *Learning multiscale sparse representations for image and video restoration.* SIAM Multiscale Modeling and Simulation, 7(1), páginas 214-241, (2008).

[22] J.M. Ogden, E.H. Adelson, J.R. Bergen y P.J. Burt. *Pyramid-based computer graphics.* RCA Engineer, 30(5), páginas 4-15, (1985).

[23] S. Masnou y J.-M. Morel. *Level lines based disocclusion.* In 5th IEEE Int. Conf. on Image Processing, Chicago, Illinois, (Octubre de 1998).

[24] T.F. Chan y J. Shen. *Mathematical models for local deterministic inpaintings.* SIAM Journal of Applied Math., 62(3), páginas 1019-1043, (2001).

[25] T.F. Chan, S.H. Kang y J. Shen. *Euler's elastica and curvature based inpainting.* SIAM Journal of Applied Math., 63(2), páginas 564-592, (2002).

[26] C. Ballester, M. Bertalmío, V. Caselles, G. Sapiro y J. Verdera. *Filling-in by joint interpolation of vector fields and gray levels.* IEEE Trans. On Image Processing, 10(8), páginas 1200-1211, (2001).

[27] S. Esedoglu y J. Shen. *Digital image inpainting by the Mumford-Shah-Euler image model.* European J. Appl. Math., vol. 13, páginas 353-370, (2002).

[28] S. Esedoglu, S. Ruuth y R. Tsai. *Threshold dynamics for high order geometric motions.* Interfaces and Free Boundaries, (2007).

[29] D. Tschumperlé. *Fast anisotropic smoothing of multi-valued images using curvature-preserving PDE's.* International Journal of Computer Vision, 68(1), páginas 65-82, (2006).

[30] D. Auroux y M. Masmoudi. *A one-shot inpainting algorithm based on the topological asymptotic analysis.* Comp. Appl. Math., vol. 25, páginas 1-17, (2006).

[31] N. Komodakis y G. Tziritas. *Image completion using efficient blief propagation via priority scheduling and dynamic pruning.* IEEE Transactions on Image Processing, 16(11), página 2649, (2007).

[32] I. Drori, D. Cohen-Or y H. Yeshurun. *Fragment-based image completion.* In Proc. SIGGRAPH'03, vol. 22(3), páginas 303-312, (2003).

[33] J. Hays y A.A. Efros. *Scene completion using millions of photographs.* Communications of the ACM, vol. 51(10), páginas 87-94, (2008).

[34] F. Catté, P.L. Lions, J.M. Morel y T. Coll. *Image selective smoothing and edge detection by nonlinear diffusion.* SIAM J. Numer. Anal., vol. 29, páginas 182-193, (1992).

[35] B. Fischl y E.L. Schwartz. *Learning an integral equation approximation to nonlinear anisotropic diffusion in image processing.* IEEE Trans. Pattern Anal. Mach. Intell., vol. 19(4),páginas 345-352, (Abril 1997).

[36] B. Fischl y E.L. Schwartz. *Adaptive nonlocar filtering: a fast alternative to anisotropic diffusion for image enhancement.* IEEE Trans. Pattern Anal. Mach. Intell., vol. 21(1), páginas 42-49, (Enero 1999).

[37] G. Gerig, O. Kübler, R. Kikinis y F.A. Jolesz *Nonlinear anisotropic filtering of MRI data.* IEEE Transactions on Medical Imaging, vol. 11(2), páginas 221-232 (Junio 1992).

[38] M. Niztberg y T. Shiota. *Nonlinear image filtering with edge and corner enhancement.* IEEE Transactions on Pattern Analysis ans Machine Intelligence, vol. 14(8), páginas 826-833, (Agosto 1992).

[39] M. Karaman, M. A. Kutay y G. Bozdagi. *An adaptive speckle suppresion filter for medical ultrasonic imaging.* IEEE transactions on medical imaging, vol. 14(2), páginas 283-292, (Junio 1995).

[40] M. J. Black y G. Sapiro. *Edges as outliers: anisotropic smoothing using local image statistics.* Lecture Notes on Computer Science, vol. 1682, páginas 259-270, (Septiembre 1999).

[41] M. J. Black, G. Sapiro, D. H. Marimont y D. Heeger. *Robust anisotropic diffusion.* IEEE Trans. Image Proccess., vol. 7(3), páginas 421-432, (Marzo 1998).

[42] A. Witkin. *Scale-Space Filtering.* Int'l Joint Conf. Artificial Intelligence, páginas 1019-1021, (1983).

[43] J. Koenderink. *The Structure of Images.* Biological Cybernetics, vol. 50, páginas 363-370, (1984).

[44] H. Scharr, M. J. Black y H. W. Haussecker. *Image Statistics and Anisotropic Diffusion.* Proceedings. Ninth IEEE International Conference on Computer Vision, vol. 2, páginas 840-847, (Octubre 2003).

Apéndice A

Manual de usuario de la aplicación *PhotoRestore*

La intencionalidad del presente manual es poder orientar a cualquier usuario en el uso de la aplicación *Android* que se ha creado. Como

A.1. Instalación

Las aplicaciones *Android*, una vez compiladas, se empaquetan en ficheros con extensión APK. Este tipo de ficheros contiene la información necesaria para realizar la instalación dentro del dispositivo.

La instalación de la aplicación se puede realizar en principio en cualquier dispositivo *Android* que posea la versión 2.2 o superior. A partir de aquí se puede hacer una diferenciación entre realizar la instalación dentro de un dispositivo real o uno emulado dentro de un ordenador.

A.1.1. Dispositivo físico *Android*

El caso más normal que se puede dar es querer instalar la aplicación dentro de un dispositivo real para hacer uso de sus funcionalidades. Para ello se va a exponer el modo de realizarlo.

1. Inicialmente se debe conectar el dispositivo *Android* a un equipo mediante un cable USB o vía Bluetooth para realizar transferencias de datos.
2. Se toma el fichero APK que se ha nombrado anteriormente y se copia dentro de la memoria del dispositivo.
3. Se desconecta de forma segura el dispositivo de la conexión que se ha utilizado para transferir los datos.
4. Mediante el "Explorador de ficheros" del que está provisto el dispositivo se va hasta la ruta donde se ha copiado el fichero APK.

5. Se pulsa encima del fichero APK para ejecutarlo.

6. Cuando se haya pulsado el fichero aparecerá una pantalla donde se pregunta si se desea instalar la aplicación y donde se avisa de los permisos que ésta necesita. En la parte de abajo de esta pantalla aparecen dos botones ("Instalar" y "Cancelar") mediante los cuales se puede iniciar la instalación o cancelarla. Se debe pulsar "Instalar".

7. El dispositivo ejecutará una serie de operaciones para instalar la aplicación. Una vez instalada aparecerá una nueva pantalla donde se mostrará el mensaje "Aplicación Instalada". Más abajo existen dos botones "Abrir", para iniciar la aplicación, y "Listo", para finalizar la instalación y realizar otras acciones con el dispositivo.

Esta es la forma más sencilla y común de instalar la aplicación en un dispositivo *Android*.

A.1.2. Emulador *Android Virtual Device*

Si se quiere ejecutar la aplicación utilizando la emulación de un dispositivo virtual se deben ejecutar también una serie de pasos.

Las siguientes instrucciones han sido ejecutadas y probadas bajo *Microsoft Windows 7*.

1. Instalar la última JDK publicada por *Oracle*.

2. Descargar los últimos paquetes del SDK de *Android* de *Google* desde "http://developer.android.com/sdk/" e instalarlo.

3. Ejecutar el "SDK Manager" y descargar e instalar la versión de *Android* donde se quiera proceder a ejecutar la aplicación. Esperar a que finalice la operación.

4. Crear un dispositivo virtual *Android* mediante la aplicación "AVD Manager", que es el gestor de máquinas virtuales *Android*. Pulsando el botón "New..." que se muestra en la imagen superior de la Figura A.1. Rellenar los campos necesarios de la ventana de diálogo, como se muestra en la imagen inferior de la Figura A.1. Y pulsar el botón "Create AVD".

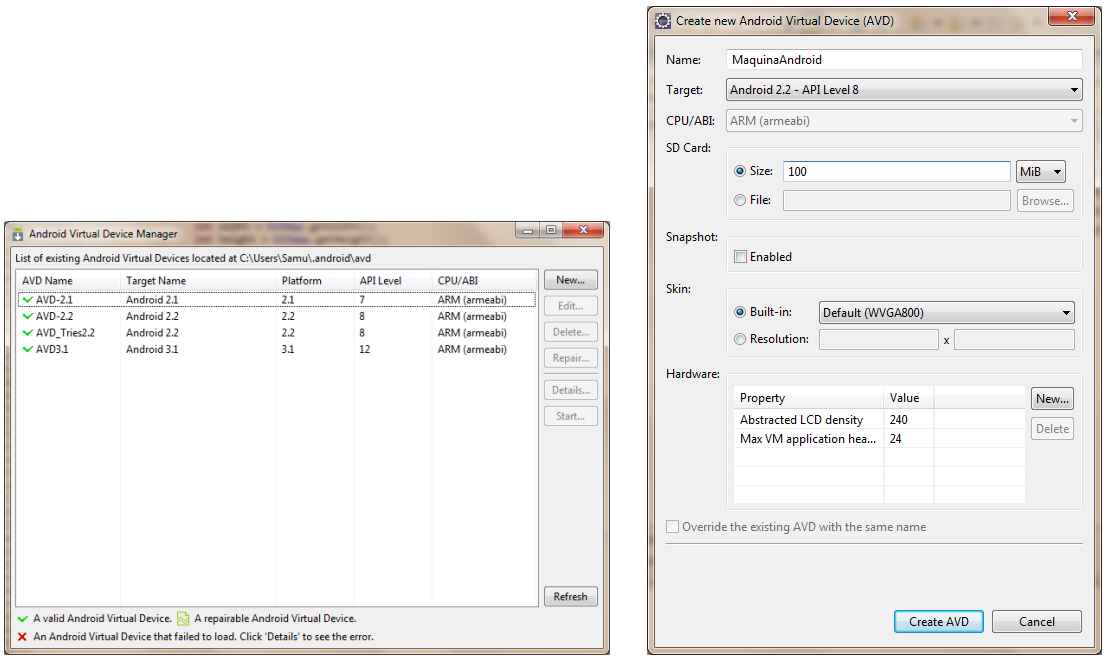


Figura A.1: Crear Dispositivo Virual *Android*

5. En el siguiente paso se debe iniciar la máquina virtual que se acaba de crear. Para ello hay que seleccionarla y después se debe pulsar el botón "Start" que aparece en la imagen superior de la Figura A.2.
   Cuando aparezca la imagen inferior de la Figura A.2 se debe pulsar el botón "Launch" y con eso se ejecutará el emulador y se creará una instancia de una máquina virtual *Android* para la versión que hayamos seleccionado. Una vez lanzada hay que esperar a que finalice de forma correcta la carga del sistema.

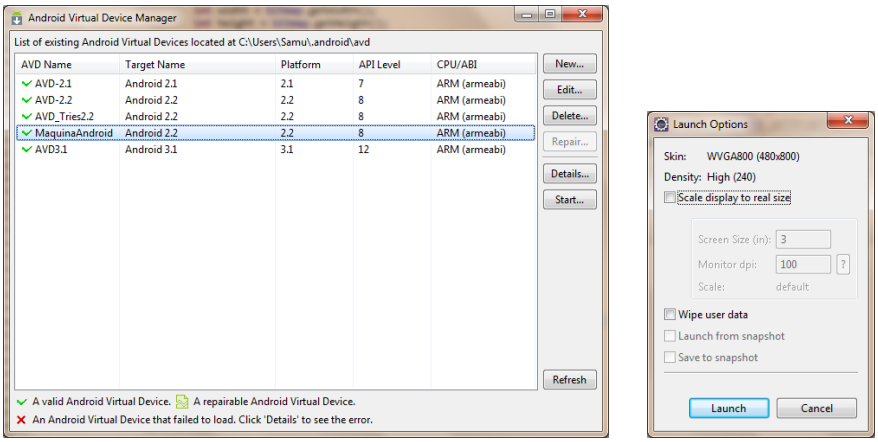


Figura A.2: Ejecutar Dispositivo Virual *Android*

6. Copiar el fichero APK dentro de la carpeta "<android-sdk-root>/plataform-tools/", donde "<android-sdk-root>" es la carpeta raíz donde se ha instalado la SDK *Android*.

7. Ejecutar una ventana de comandos de Windows o *cmd*. Cambiar la carpeta a la dirección donde acaba de ser copiado el APK ("<android-sdk-root>/plataform-tools/") utilizando el comando "cd".

8. Una vez dentro de la carpeta introducir el comando "adb install *.apk", donde el * hace referencia al nombre del fichero APK que se desea instalar.

9. Si la instalación se realiza de forma correcta se recibirá un mensaje "Success". Con esto ya puede dirigirse al menú de aplicaciones del dispositivo virtual *Android* que se está ejecutando y dentro encontrará la aplicación instalada.

En caso de que obtengan mensajes de error, por favor, lea otra vez el tutorial. Verifique que ha copiado el fichero APK en la carpeta indicada y que no se da ningún error al escribir el nombre del fichero (el programa distingue las letras mayúsculas de las minúsculas).

A.2. Uso de la aplicación

En este apartado se quiere hacer una explicación de todas las funcionalidades programadas en la aplicación *Android*. Esta aplicación ha sido denominada *PhotoRestore* debido a que la utilidad que se le ha querido dar es la de resturar fotografías ruidosas o dañadas.

La aplicación se compone de varias pantallas. A continuación, se va a desgranar el funcionamiento de cada uno de los botones.

A.2.1. Pantalla principal

Cuando se ejecuta la aplicación *PhotoRestore* se muestra la imagen (a) de la Figura A.3.

a b c d

Figura A.3: **Pantalla principal**: (a) Inicio, (b) Menú de -, (c) Ventana de ayuda de -, y (d) Acerca de la aplicación.

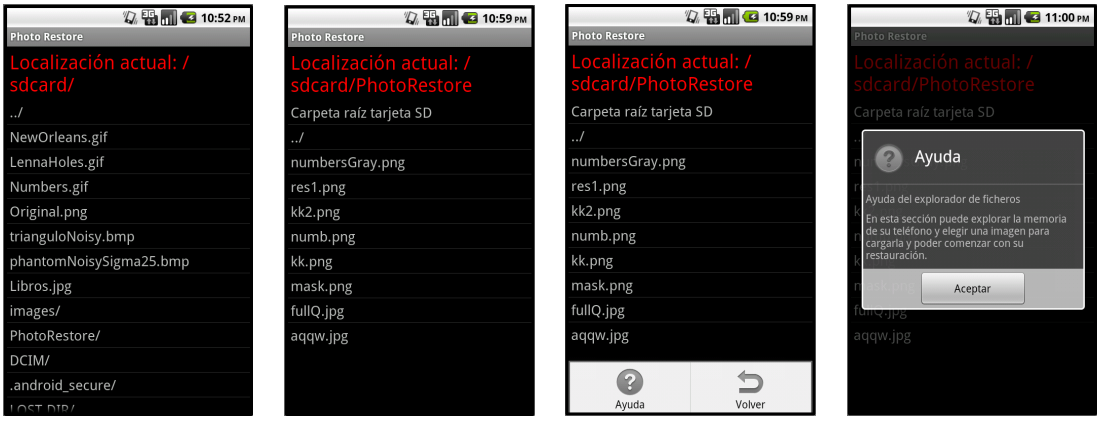
En la pantalla se puede apreciar un título y dos botones: "Abrir Archivo" y "Acerca

de..".
El primero de los botones lanza una nueva pantalla que se muestra en la Figura A.4(a). Por su parte el botón "Acerca de..." mostrará la ventana de diálogo que aparece en la Figura A.3(d).

Si se pulsa el botón *Menú* del dispositivo *Android*, en ese caso se mostrará una pantalla similar a la ilustración (b) de la Figura A.3. Como se muestra, se pueden encontrar dos botones "Ayuda" y "Salir".
El botón "Ayuda" lanza una ventana similar a la mostrada en A.3(c).
El botón "Salir" lanzará una pequeña ventana de consulta. Dentro de ella se podrá confirmar o desmentir la acción de cerrar la aplicación.

A.2.2. Explorador de ficheros

Esta pantalla dará la opción al usuario de explorar la unidad de almacenamiento de su dispositivo *Android* y encontrar dentro de ella la imagen que desee restaurar.



a b c d

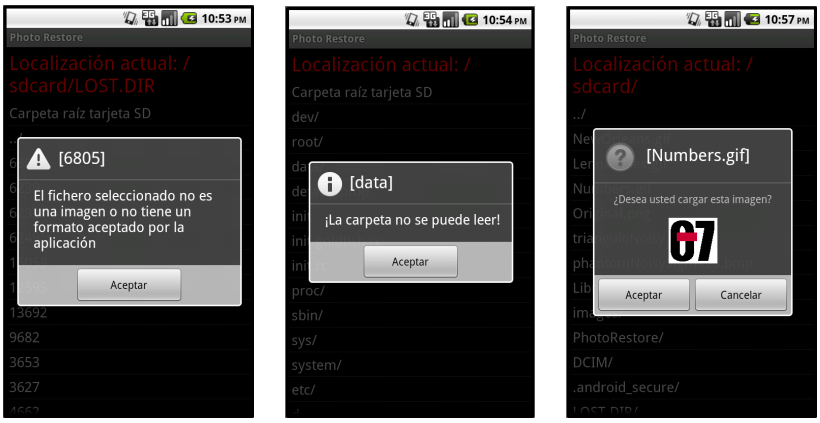
Figura A.4: **Explorador de archivos**: (a) Inicio, (b) Una carpeta en medio de la navegación, (c) Menú del - y (d) Ventana de ayuda del -.

Inicialmente se mostrará la Figura A.4(a). Esta ilustración presenta una cabecera en rojo donde se indica la ubicación actual del usuario y también presenta una lista de ficheros y carpetas de la ruta que se encuentra seleccionada en el instante.
Si el usuario comienza a navegar entre las distintas carpetas, dejará de encontrarse localizado en la carpeta raíz de la tarjeta SD (el almacenamiento externo del dispositivo), entonces se mostrará una entrada con el lema "Carpeta raíz Tarjeta SD", como se muestra en la Figura A.4(b). Pulsar esta entrada enviará al usuario a la pantalla inicial y volverá a mostrar el listado de ficheros que se encuentran dentro de la ruta inicial.
Otra entrada generalmente corresponderá con un símbolo "../". Este símbolo es para acceder a la carpeta donde se aloja la ruta actual dentro del árbol de directorios.
Cualquier entrada que finaliza con una barra inclinada "/", excepto "../", indica que eso es una carpeta.
El resto de entradas generalmente serán archivos o ficheros que se pueden tratar de abrir. Lo más sencillo para distinguir las imágenes será inspeccionar las extensiones de los ficheros.

Esta pantalla también tiene asignadas funcionalidades al pulsar el botón *Menú* del dispositivo *Android*, en ese caso se mostrará una pantalla similar a la ilustración (c) de la Figura A.4. Como se muestra, se pueden encontrar dos botones "Ayuda" y "Volver".
El botón "Ayuda" lanza una ventana similar a la mostrada en A.4(d).
El botón "Volver" devolverá al usuario a la pantalla inicial, A.3(a).

Cuando una carpeta sea seleccionada podrán darse dos opciones: abrir la carpeta y mostrar su contenido o mostrar un mensaje de error como el de la Figura A.5(b) debido a que la carpeta no es legible.

En el caso de que se seleccione un fichero o archivo existen tres opciones las cuales se muestran en la Figura A.5.
La primera de las opciones, mostrada en la ilustración (a) de la Figura A.5 se dará en el caso de que el usuario seleccione un fichero que no sea una imagen o sí que sea un fichero de imagen pero no esté soportado por la aplicación (por ejemplo, el formato TIFF).
La segunda de las opciones, la cual se ilustra en la Figura A.5(b), se mostrará cuando el usuario seleccione una imagen, fichero o carpeta el cual no se pueda leer.
Finalmente, cuando el usuario seleccione un fichero que sí sea legible y sobre el que pueda aplicar las funcionalidades que se dan en la aplicación, se mostrará la ventana de diálogo como aparece en la Figura A.5(c). Esta ventana incluye una vista previa de la imagen, el nombre de la imagen y, en ella, también se solicita la confirmación de carga de la imagen. Cuando el usuario pulse el botón "Aceptar" será dirigido a una pantalla como en la mostrada en la Figura A.6(a), la cual será descrita a en el siguiente apartado. En caso contrario, que pulse "Cancelar" el usuario podrá continuar utilizando el "Explorador de Ficheros" hasta que encuentre la imagen deseada.



a b c

Figura A.5: **Selección del archivo**: (a) Fichero seleccionado no es imagen, (b) Carpeta no legible y (c) Carga de una imagen.

A.2.3. Menú de imagen

La imagen que se presentará cuando el usuario cargue en la aplicación una imagen para restaurar será la mostrada en la Figura A.6(a).

Esta pantalla tiene asignadas funcionalidades al pulsar el botón *Menú* del dispositivo

a b c d

Figura A.6: **Vista previa de la imagen**: (a) Inicio, (b) Menú de -, (c) Ventana de selección de modalidad de *inpainting* y (d) Ventana de ayuda de -.

Android. Dentro de esta sección se esconde el grueso de las acciones que se pueden realizar desde este punto. El menú mostrará una pantalla similar a la ilustración (b) de la Figura A.6. En dicha imagen pueden distinguirse cinco opciones diferentes "Inpaint", "Reducir ruido", "Guardar", "Ayuda" y "Cargar otra imagen".
Seleccionar el botón "Inpaint" mostrará una ventana de diálogo como la de la Figura A.6(c). Dentro de esta ventana de diálogo el usuario debe seleccionar el tipo de selección de máscara para *inpainting* que desea realizar.
El botón "Reducir ruido" enviará al usuario a la funcionalidad para reducir el ruido de la imagen mediante la técnica del *filtrado de difusión anisótropa*.
La opción "Guardar" implementa la funcionalidad de almacenar como un archivo la imagen que se muestra en la vista previa. Esta funcionalidad será descrita más adelante.
El botón "Ayuda" lanza una ventana similar a la mostrada en A.6(d).
El botón "Cargar otra imagen" devolverá al usuario al "Explorador de archivos", A.4(a), para que pueda cargar otra imagen.

A.2.3.1. Guardar imagen

Tras una restauración satisfactoria mediante las funcionalidades implementadas el usuario será devuelto a esta pantalla. En ella se mostrará la imagen restaurada, la cual podrá guardar utilizando la opción "Guardar" del menú expuesta anteriormente. Las imágenes de la Figura A.7 muestran las opciones para guardar la imagen.

En la imagen A.7(a) se puede observar la ventana de diálogo donde se deberán introducir los datos necesarios. Estos datos serán el nombre con el cual se quiere almacenar el fichero y el formato con el cual se desea guardar el fichero.

El usuario deberá introducir el nombre con el cual desea que se almacene el fichero. Por otra parte, la imagen podrá ser almacenada con dos formatos distintos sin pérdida de calidad: JPEG y PNG.

Una vez rellenados los campos el usuario deberá pulsar el botón "Aceptar" para alma-

a b

Figura A.7: **Guardar la imagen**: (a) Ventana de guardado de la imagen y (b) campo de selección del formato de archivo.

cenar la imagen. Si ésta se almacena de forma correcta, el usuario recibirá un mensaje donde se le indicará que el fichero se ha guardado de forma satisfactoria. La imagen, una vez almacenada se podrá encontrar dentro del directorio "/sdcard/PhotoRestore/". Dentro de este directorio se almacenarán todas las imágenes mediante este procedimiento.
En el caso contrario, que quiera desechar el almacenamiento sólo debe pulsar el botón "Cancelar" y será devuelto a la vista previa de la imagen, A.6(a).

A.2.4. *Inpainting*

Dependiendo de la funcionalidad de *inpainting* que seleccione el usuario en la Figura A.6(c), se le mostrarán dos funcionalidades distintas para realizar la selección de la máscara de *inpainting*:

A.2.4.1. Selección manual

En la selección manual el usuario deberá dibujar con su dedo sobre la superficie de la imagen para seleccionar la región que desee que sea la zona a restaurar. En la imagen (a) de la Figura A.8 se muestra la sugerencia que se le realiza inicialmente al usuario para que seleccione la máscara.
En las imágenes (b) y (c) de la misma Figura se muestra el menú que se mostrará cuando se pulse la tecla *Menú* del dispositivo *Android* y la ayuda que se puede desplegar sobre el menú de la pantalla mediante la opción "Ayuda".

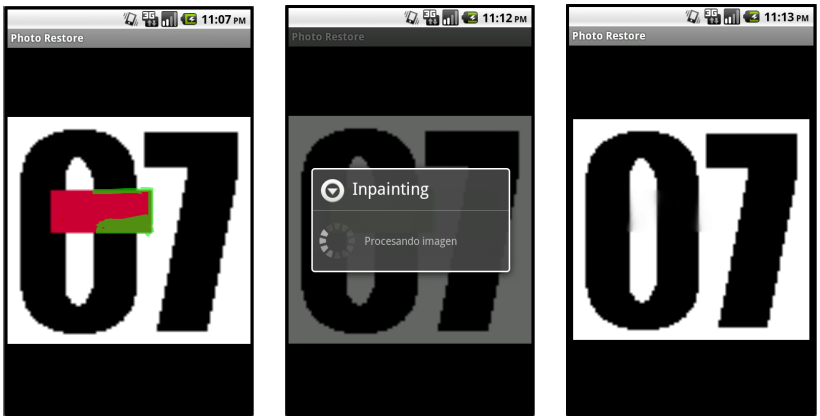
El resto de botones que se muestran en el menú de la pantalla son "Aceptar", que lanza el proceso de *inpainting* con la máscara seleccionada; "Reiniciar selección", lo cual reinicia la máscara dejando el lienzo donde dibujarla tal como se encontraba inicialmente; y "Volver", que envía al usuario de vuelta a la pantalla del menú de la imagen sin haber realizado ninguna modificación en la imagen, como la que se muestra en la Figura A.6(a).

Como ya se ha dicho, el usuario deberá dibujar la máscara de *inpainting* sobre la imagen, deslizando el dedo. Existe una muestra de la realización de este proceso, el cual

a b c

Figura A.8: ***Inpainting* mediante selección manual**: (a) Mensaje inicial, (b) Menú de - y (c) Ventana de ayuda de -.

se muestra en A.9(a).



a b c

Figura A.9: ***Inpainting* mediante selección manual**: (a) Instantánea del proceso de selección de máscara, (b) pantalla mostrada durante el procesado y (c) resultado de -.

Una vez el usuario haya finalizado la selección de la máscara debe acudir al menú y pulsar la opción "Aceptar". Tras ello se mostrará durante el tiempo de procesado una imagen similar a A.9(b). Cuando el procesado haya finalizado se mostrará la imagen restaurada dentro de una pantalla similar a la de la Figura A.6, incluyendo las mismas opciones que en estas imágenes se muestran.

A.2.4.2. Selección de color

En la selección mediante color el usuario pulsará con su dedo sobre la superficie que contenga el color que ocluya la información de la región que se desea restaurar. En la imagen (a) de la Figura A.10 se muestra la sugerencia que se le realiza inicialmente al

usuario para que seleccione la máscara. El usuario tras haber seleccionado un color podrá seguir pulsando para seleccionar más colores. Esto se debe a que en algunas imágenes podrían darse pequeños degradados de color en la zona ocluída y, entonces habría que seleccionar más de un color.
En las imágenes (b) y (c) de la misma Figura se muestra el menú que se mostrará cuando se pulse la tecla *Menú* del dispositivo *Android* y la ayuda que se puede desplegar sobre el menú de la pantalla mediante la opción "Ayuda".

El resto de botones que se muestran en el menú de la pantalla son iguales a los de la sección anterior: "Aceptar", que lanza el proceso de *inpainting* con la máscara seleccionada; "Reiniciar selección", lo cual reinicia la máscara dejando el lienzo donde dibujarla tal como se encontraba inicialmente; y "Volver", que envía al usuario de vuelta a la pantalla del menú de la imagen sin haber realizado ninguna modificación en la imagen, como la que se muestra en la Figura A.6(a).

a b c

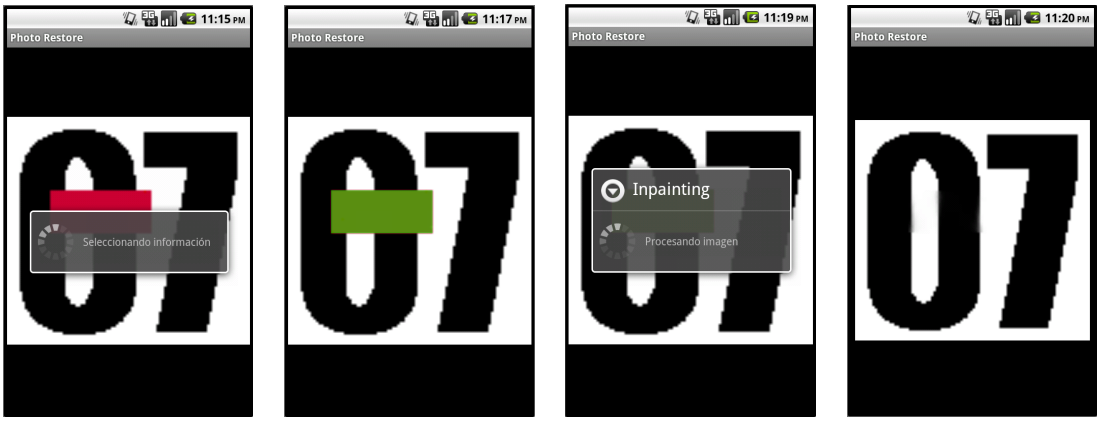
Figura A.10: ***Inpainting* mediante selección de color**: (a) Mensaje inicial, (b) Menú de - y (c) Ventana de ayuda de -.

Como ya se ha dicho, el usuario deberá seleccionar el color o colores de la máscara de *inpainting* sobre la superficie de la imagen, pulsando con su dedo en los puntos deseados. En el momento en que se pulsa un punto, se lanza una ventana de bloqueo para procesado como se muestra en A.11(a).
Cuando la ventana desaparece se muestra ya la imagen con la región seleccionada con un halo verde como en la ilustración (b) de la Figura A.11.

Una vez realizada la selección de la máscara debe acudir al menú y pulsar la opción "Aceptar". Tras ello se mostrará durante el tiempo de procesado una imagen similar a A.11(c). Cuando el procesado haya finalizado se mostrará la imagen restaurada dentro de una pantalla similar a la de la Figura A.11(d), incluyendo las mismas opciones que se muestran en las imágenes de la Figura A.6.

A.2.5. Reducción de ruido

En este apartado se va a mostrar el uso de la funcionalidad de reducción de ruido en la imagen. A esta funcionalidad se accede desde el menú de imagen mostrado en la ilustración



a b c d

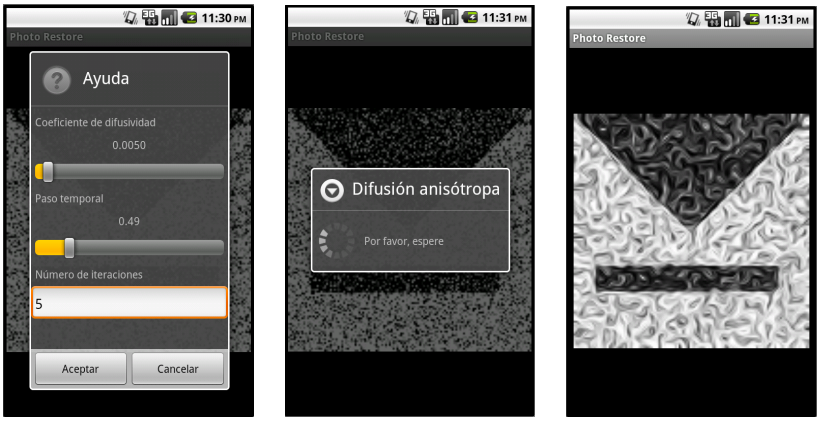
Figura A.11: ***Inpainting* mediante selección de color**: (a) Instantánea del proceso de selección de máscara, (b) ejemplo de selección de máscara de -, (c) pantalla mostrada durante el procesado y (d) resultado de -.

A.6(b) pulsando la opción *Reducir ruido*.
En la Figura A.12 se muestran las instantáneas de la pantalla inicial (a), el menú que aprecer al pulsar el botón *Menú* del dispositivo *Android* (b) y la ayuda que se presta para esta funcionalidad (c).

a b c

Figura A.12: **Reducción de ruido**: (a) Pantalla inicial, (b) Menú de - y (c) Ventana de ayuda de -.

Dentro del menú (Figura A.12(b)) se muestran varias opciones.
La opción "Ayuda" muestra la ayuda de la imagen (c) de la Figura A.12. Dentro de esta ayuda se pueden
La opción "Volver" enviará al usuario de vuelta a la pantalla correspondiente a la ilustración A.6(a) sin realizar modificación ninguna a la imagen.
Por último, el botón "Establecer parámetros" lanza la ventana mostrada en la Figura A.13(a).



a b c

Figura A.13: **Reducción de ruido**: (a) Pantalla de diálogo donde seleccionar los valores para los parámetros del algoritmo, (b) pantalla mostrada durante el procesado y (c) resultado de -.

En esta ventana de selección de parámetros se deben introducir los valores para el coeficiente de difusividad, el paso temporal y también el número de iteraciones que se ejecutarán en el método de *filtrado de difusión anisótropa*.
Los dos primeros parámetros se seleccionan mediante unas barras de selección. Encima de estas barras de selección se muestra el valor del parámetro con el cual se realizará la ejecución cuando se pulse el botón "Aceptar" de la misma ventana.

El último de los parámetros, el número de iteraciones, debe ser introducido mediante un valor numérico en el campo que le corresponde. Los valores permitidos se encuentran en el intervalo desde 1 hasta 100.

En cualquier momomento el usuario puede pulsar "Cancelar" para anular la introducción de parámetros y volver a la pantalla anterior A.12(a).

Tras introducir correctamente todos los parámetros el usuario pulsará el botón "Aceptar" para realizar la restauración de la imagen. En ese momento, comenzará el procesado de restauración mostrando durante todo el proceso la imagen A.13(b).
Cuando finalice el proceso de *filtrado de difusión anisótropa* al usuario le será mostrada una imagen similar a A.13(c), donde el usuario tendrá las mimas opciones de uso que se presentan en las imágenes de la Figura A.6.

Apéndice B

Especificaciones técnicas

En este capítulo se detallan las especificaciones técnicas de los dispositivos utilizados en los experimentos y pruebas realizados en este trabajo.

B.1. Ordenador portátil

El ordenador con el que se han realizado las pruebas es un Lenovo ThinkPad Edge cuyo modelo corresponde con la numeración NV13BSP.
Las principales especificaciones técnicas son las siguientes:

Procesador:	Intel Core i3-380UM (1,33 Ghz / 3 Mb. Cache)
Memoria:	4 GB. DDR3 a 1333 Mhz.
Disco Duro:	500 GB. a 5400 rpm.
Sistema Operativo:	Windows 7 Home Premium (64 bits).
Tarjeta Gráfica:	Intel GMA HD (integrada).

B.2. Tablet

El tablet que posee el autor es un tablet bastante desfasado en el momento de presentar este trabajo. Se trata de un Archos 70 Internet Tablet.
Las principales especificaciones técnicas del dispositivo portátil son las siguientes:

Procesador:	ARM Cortex A8 1 GHz con DSP.
Memoria:	256 MB.
Disco Duro:	8 GB memoria flash.
Sistema Operativo:	Android 2.2.
Tarjeta Gráfica:	Acelerador gráfico 3D OpenGL ES 2.0.

Apéndice C

Implementación Matlab

C.1. Filtrado de difusion anisótropa

C.1.1. Filtro de Perona-Malik

Listing C.1: **anisoFilt.m**

```
function [ filtered_image ] = anisoFilt( original_image, delta_t,
    K, func_diff, num_iters )
%ANISOT_FILT Filtrado anisótropo 2D
% Input:
%  · original_image — Imagen original con ruido
%  · delta_t — Variación temporal entre los pasos
%  · K — Constante del coeficiente de difusión
%  · fun_diff — Cadena donde se introdude la función de difusión
   a
%  utilizar
%  · num_iters — Número de veces que se iterará el algoritmo
% Output:
%  · filtered_image — Imagen procesada mediante el filtrado
   anisótropo

%  Se verifica que la variación temporal se encuentra dentro de
   la región
%  de estabilidad
   if ( (delta_t <= 0) || (delta_t > (1/7)) )
      return;
   end

%  Se dan los valores para las distancias entre los píxeles en la
%  dimensión x y en la dimensión y
   delta_x = 1;
   delta_y = 1;

%   Se declaran las máscaras para filtrar la imagen para cada
   uno de los
%   flujos
   mE = [0 0 0;
```

```
        0 -1 1;
        0 0 0];
    mO = [0 0 0;
        -1 1 0;
        0 0 0];
    mN = [0 0 0;
        0 -1 0;
        0 1 0];
    mS = [0 -1 0;
        0 1 0;
        0 0 0];

%     Se declaran las máscaras para calcular los gradientes
    mDE = [0 0 -1;
        0 0 0;
        0 0 1]/2;
    mDO = [-1 0 0;
        0 0 0;
        1 0 0]/2;
    mDN = [0 0 0;
        0 0 0;
        -1 0 1]/2;
    mDS = [-1 0 1;
        0 0 0;
        0 0 0]/2;

%    Se establece el estado inicial de la ecuación de difusión
    filtered_image = original_image;

    for it = 1:num_iters

%       Se calculan los valores de la variación en los flujos.
    Primero en
%       las direcciones ortogonales y luego en las diagonales
        deltaE = filter2(mE, filtered_image);
        deltaO = filter2(mO, filtered_image);
        deltaN = filter2(mN, filtered_image);
        deltaS = filter2(mS, filtered_image);
        deltaDE = filter2(mDE, filtered_image);
        deltaDO = filter2(mDO, filtered_image);
        deltaDN = filter2(mDN, filtered_image);
        deltaDS = filter2(mDS, filtered_image);

%       Se calcula el el gradiente para cada flujo
        gE = sqrt( (deltaE).^2/delta_x + ( (deltaDE)/2 ).^2/delta_y
            );
        gO = sqrt( (deltaO).^2/delta_x + ( (deltaDO)/2 ).^2/delta_y
            );
        gN = sqrt( (deltaN).^2/delta_x + ( (deltaDN)/2 ).^2/delta_y
            );
        gS = sqrt( (deltaS).^2/delta_x + ( (deltaDS)/2 ).^2/delta_y
            );
```

```
%       Se calcula el valor de la función de difusión para cada
    flujo
      cE = feval(func_diff, gE, K);
      cO = feval(func_diff, gO, K);
      cN = feval(func_diff, gN, K);
      cS = feval(func_diff, gS, K);

%       Se calcula el flujo en cada uno de los sentidos de las
%       direcciones
      fE = 1/(delta_x).^2*( cE.*deltaE );
      fO = 1/(delta_x).^2*( cO.*deltaO );
      fN = 1/(delta_y).^2*( cN.*deltaN );
      fS = 1/(delta_y).^2*( cS.*deltaS );

%       Se calcula la solución de la iteración
      filtered_image = filtered_image + delta_t*( fE - fO + fN -
          fS );

   end

end
```

Listing C.2: **c1.m**

```
function [ out ] = c1( x, k )
%C1 Summary of this function goes here
%  Detailed explanation goes here

   out = exp(-(x/k).^2);

end
```

Listing C.3: **c2.m**

```
function [ out ] = c2( x, k )
%C2 Summary of this function goes here
%  Detailed explanation goes here

   out = 1./(1 + (x/k).^2);

end
```

Listing C.4: **demo.m**

```
clear all; close all;

delta_t = 1/10;
K = 2;
func_diff = 'c1';
num_iters = [10 25 50 100];

I = double(phantom(256));

% Se le añade ruido a la imagen
J = imnoise(I, 'salt & pepper', 0.1);
```

```
I_filt = zeros(size(J,1), size(J,2), length(num_iters));

for i = 1:length(num_iters)
   I_filt(:,:,i) = anisoFilt( J, delta_t, K, func_diff, num_iters
      (i) );
end

figure(1);
subplot(2,3,1);
imagesc(I);
title('Imagen original');
colormap(gray);

subplot(2,3,2);
imagesc(J);
title('Imagen con ruido "salt & pepper"');
colormap(gray);

subplot(2,3,3);
imagesc(I_filt(:,:,1));
colormap(gray);
title(['Imagen filtrada con K=', num2str(K), ' y iters=', num2str
    (num_iters(1))]);

subplot(2,3,4);
imagesc(I_filt(:,:,2));
colormap(gray);
title(['Imagen filtrada con K=', num2str(K), ' y iters=', num2str
    (num_iters(2))]);

subplot(2,3,5);
imagesc(I_filt(:,:,3));
colormap(gray);
title(['Imagen filtrada con K=', num2str(K), ' y iters=', num2str
    (num_iters(3))]);

subplot(2,3,6);
imagesc(I_filt(:,:,4));
colormap(gray);
title(['Imagen filtrada con K=', num2str(K), ' y iters=', num2str
    (num_iters(4))]);
```

C.1.2. Filtro de Weickert

Listing C.5: **anisotropicDiffusion.m**

```
function [ imagen ] = anisotropicDiffusion( imagen, maxIters, ...
                          time_step, k, debugMode )
%ANISOTROPICDIFFUSION Realiza la difusión anisótropa de la
   imagen según el
%algoritmo propuesto por Weickert
% Entrada:
%  · imagen — Imagen a la que se le va a eliminar el ruido
%  · maxIters — Número máximo de iteraciones del algoritmo
%  · time_step — Paso temporal que va a utilizar el algoritmo
   para avanzar
%  en la ecuación de difusión
%  · k — Coeficiente de difusividad para calcular la pertenencia
   a un
%  borde
%  · debugMode — Variable booleana para ejecutar en modo
   depuración, con
%  visualización intermedia de pasos
% Salida:
%  · imagen — Imagen con el ruido ya eliminado

   if (~exist('maxIters', 'var')) || (maxIters <= 0)
      maxIters = 25;
   end
   if (~exist('time_step', 'var')) || (time_step <= 0)
      time_step = .1;
   end
   if (~exist('k', 'var') || (k < 0) )
      k = 0.01;
   end
   if (~exist('debugMode', 'var') || isempty(debugMode))
      debugMode = 0;
   end

   % Creación de variables utilizadas a lo largo de todo el
      algoritmo
   original_imagen = imagen;

   sizeGaussian = 5;
   sigmaGaussian = 1;

   for idx = 1:maxIters

      if (size(imagen,3) > 1)
         gray_imagen = rgb2gray(imagen);
      else
         gray_imagen = imagen;
      end

      % Cálculo del tensor de estructura de cada punto de la
         imagen
      [ Jxx, Jxy, Jyy ] = tensorEstructura(gray_imagen, ...
```

```
            sizeGaussian, sigmaGaussian);

        % Cálculo de los autovalores del tensor de estructura de
            cada
        % uno de los puntos de la imagen
        [ mu1, mu2, vx, vy, ux, uy] = ...
            autovaloresAutovectores( Jxx, Jxy, Jyy );

        % Cálculo del tensor de difusión
        [Dxx, Dxy, Dyy] = tensorDifusion( mu1, mu2, ...
            vx, vy, ux, uy, k, debugMode );

        % Difusión de la imagenn según el tensor de difusión
        imagen = esquemaDifusion(imagen, Dxx, Dxy, Dyy, time_step);

        % Representación de la imagen filtrada en caso de que se
            encuentre
        % ejecutando en modo de depuración
        if (debugMode == 1)
            disp(['Iteración numero: ' num2str(idx)])
            figure(115);
            subplot(121);
            imshow(original_imagen, ...
                [min(original_imagen(:)) max(original_imagen(:))]);
            subplot(122);
            imshow(imagen, ...
                [min(original_imagen(:)) max(original_imagen(:))]);
        end

    end

end
```

Listing C.6: **tensorEstructura.m**

```
function [ Jxx, Jxy, Jyy ] = tensorEstructura( imagen, sizeG,
    sigmaG )
%TENSOR_ESTRUCTURA Calcula las componentes del tensor de
    estructura
% Entrada:
%  · imagen — Imagen de la cual hay que calcular el tensor de
    estructura
%  · sizeG — Tamaño del filtro Gaussiano a utilizar
%  · sigmaG — Desviación típica del filtro Gaussiano a utilizar
% Salida:
%  · Jxx, Jxy, Jyy — Componentes del tensor de estructura

    filtGaussian1D = gaussian1D(sizeG, 0, sigmaG);

    imagenFiltrada = imfilter( ...
        imfilter(imagen, filtGaussian1D, 'same', 'conv', 'replicate
            '), ...
        filtGaussian1D', 'same', 'conv', 'replicate');

    Dx = derivada(imagenFiltrada, 'x');
```

```
   Dy = derivada(imagenFiltrada, 'y');

   Jxx = Dx.^2;
   Jxy = Dx.*Dy;
   Jyy = Dy.^2;

   Jxx = imfilter( ...
      imfilter(Jxx, filtGaussian1D, 'same', 'conv', 'replicate'),
          ...
      filtGaussian1D', 'same', 'conv', 'replicate');
   Jxy = imfilter( ...
      imfilter(Jxy, filtGaussian1D, 'same', 'conv', 'replicate'),
          ...
      filtGaussian1D', 'same', 'conv', 'replicate');
   Jyy = imfilter( ...
      imfilter(Jyy, filtGaussian1D, 'same', 'conv', 'replicate'),
          ...
      filtGaussian1D', 'same', 'conv', 'replicate');

end
```

Listing C.7: **gaussian1D.m**

```
function [ H ] = gaussian1D( siz, mu, sigma )
%GAUSSIAN1D Crea un filtro Gaussiano en una dimensión mediante
   los
%siguientes parámetros
% Entrada:
%  · siz — Longitud del filtro Gaussiano
%  · mu — Media del filtro Gaussiano
%  · sigma — Desviación típica del filtro Gaussiano
% Salida:
%  · H — Gaussian filter 1D as row-vector

   x = -floor(siz/2):floor(siz/2);
   H = exp( -( ( x - mu ).^2 / (2*sigma^2) ) );
% Normalización
   H = H(:)'/sum( H(:) );

end
```

Listing C.8: **derivada.m**

```
function [ derivada ] = derivada( imagen, direccion )
%DERIVADAS Calcula la derivada respecto de la dirección indicada
% Entrada:
%  · imagen — Imagen que a la que se va a aplicar el filtro de
   derivada
%  · direccion — Dirección respecto de la cual se va a alicar la
   derivada
% Salida:
%  · Valor de la derivada en los puntos de la imagen

   if ( ~exist('direccion', 'var') )
```

```
10      disp('Debe especificar la dirección en la cual quiere
            calcular la derivada');
        return;
    end

    if (direccion == 'x')
15      H = [3 10 3;
             0  0 0;
            -3 -10 -3]/32;
    elseif (direccion == 'y')
        H = [3 0 -3;
20         10 0 -10;
            3 0 -3]/32;
    else
       disp('La dirección en la cual quiere calcular la derivada
           no está disponible');
        return;
25  end

    derivada = imfilter(imagen, H, 'same', 'conv', 'replicate');

end
```

Listing C.9: **autovaloresAutovectores.m**

```
function [ mu1, mu2, vx, vy, ux, uy ] = ...
    autovaloresAutovectores( Jxx, Jxy, Jyy )
%AUTOVALORES_AUTOVECTORES Realiza el cómputo de los autovalores
    y
%autovectores del tensor de estructura a partir de sus
    componentes
5 % Entradas:
%  · Jxx, Jxy, Jyy - Componentes del tensor de estructura
% Salidas:
%  · mu1, mu2 - Autovalores del tensor de estructura
%  · vx, vy, ux, uy - Componentes de los autovectores u y v
10
    % Cálculo del discriminante
    disc = sqrt( (Jxx - Jyy).^2 + 4*Jxy.^2 );

    % Se asume el valor de ux y se calcula el de uy
15  ux = 2*Jxy;
    uy = Jyy - Jxx + disc;

    % Normalización de los autovectores
    mag = sqrt( ux.^2 + uy.^2 );
20  idx = (mag ~= 0 );
    ux(idx) = ux(idx)./mag(idx);
    uy(idx) = uy(idx)./mag(idx);

    % Los autovectores v y u son ortogonales
25  vx = -uy;
    vy = ux;

    % Cálculo de los autovalores
```

```
   mu1 = (Jxx + Jyy + disc)/2;
   mu2 = (Jxx + Jyy - disc)/2;

end
```

Listing C.10: **tensorDifusion.m**

```
function [ Dxx, Dxy, Dyy ] = tensorDifusion( mu1, mu2,...
                                    vx, vy, ux, uy, ...
                                    k, debugMode)
%TENSOR_DIFUSION Calcula el tensor de difusión
% Entrada:
%  · mu1, mu2 - Autovalores obtenidos a partir del tensor de
   estructura
%  · vx, vy, ux, uy - Autovectores correspondientes a los
   autovalores mu1
%  y mu2
%  · k - Coeficiente de difusividad
%  · debugMode - Variable booleana del modo de depuración
% Salida:
%  · Dxx, Dxy, Dyy - Componentes del tensor de difusión


   if (~exist('debugMode', 'var'))
      debugMode = 0;
   end

%% EDGE-ENHANCING DIFFUSION
   C = 3.31488;
   m = 4;
   if ( k == 0 )
      k = 2.^m;
   end

   mu1 = mu1./max(mu1(:));

   lambda1 = ones(size(mu1));
%   lambda2 = ones(size(mu1));

   exponential = exp( -(C)./(mu1./k).^m );
%   exponential = exponential./max(exponential(:));
   lambda2 = ones(size(mu1)) - exponential;

   % En caso de que la zona sea isótropa se acelera el proceso
      de filtrado
   ptosPlanos = find( (lambda1 == 1) .* (lambda2 > .9) );
   lambda1(ptosPlanos) = 2;
   lambda2(ptosPlanos) = 2;

%% Mostrando el tensor de difusión

   if (debugMode)
      figure(111);
%      subplot(121);
%      imshow(lambda1, [ ]);
```

```
%       colorbar;
%       subplot(122);
        imshow(lambda2, [ ]);
        colorbar;
        title('Tensor de difusión');
        pause;
    end

    % Se aplica el filtro anisótropo definiendo el tensor de
       difusión

    Dxx = lambda1.*vx.^2 + lambda2.*ux.^2;
    Dxy = lambda1.*vx.*vy + lambda2.*ux.*uy;
    Dyy = lambda1.*vy.^2 + lambda2.*uy.^2;

end
```

Listing C.11: **esquemaDifusion.m**

```
function [ u ] = esquemaDifusion( u, Dxx, Dxy, Dyy, dt )
%ESQUEMA_DIFUSION
% Input:
%  · u — Imagen a aplicar la difusión
%  · Dxx — Segunda derivada espacial respecto a x dos veces
%  · Dxy — Segunda derivada espacial respecto a x e y una vez
   cada una
%  · Dyy — Segunda derivada espacial respecto a y dos veces
%  · dt — Paso temporal
% Output:
%  · u — Imagen difundida

%% Esquema semi—implícito
    %% Creación de los índices de orientación
    [N1,N2,N3] = size(u);
    iR = [2:N1,N1];
    iL = [1,1:N1—1];
    iU = [2:N2,N2];
    iD = [1,1:N2—1];

    %% Asignar variables a, b y c
    a = Dxx;
    b = Dxy; abs_b = abs(b);
    c = Dyy;

    %% Calcular valores de la malla
    A1 = ( abs_b(iL, iU) — b(iL, iU) + abs_b — b )/4;
    A2 = ( c(:, iU) + c — abs_b(:, iU) — abs_b )/2;
    A3 = ( abs_b(iR, iU) + b(iR, iU) + abs_b + b )/4;
    A4 = ( a(iL, :) + a — abs_b(iL, :) — abs_b )/2;
    A5 = ( — ( a(iL, :) + 2*a + a(iR, :) ) + ...
          + ( abs_b(iL, :) + abs_b(iR, :) + abs_b(:, iD) + abs_b
              (:, iU) + 2*abs_b ) + ...
          — ( c(:, iD) + 2*c + c(:, iU) ) ...
        )/2 + ...
```

```
        - ( abs_b(iL, iU) - b(iL, iU) + abs_b(iR, iU) + b(iR,
            iU) + ...
            abs_b(iL, iD) + b(iL, iD) + abs_b(iR, iD) - b(iR, iD
                ) ...
    )/4;

  A6 = ( a(iR, :) + a - abs_b(iR, :) - abs_b )/2;
  A7 = ( abs_b(iL, iD) + b(iL, iD) + abs_b + b )/4;
  A8 = ( c(:, iD) + c - abs_b(:, iD) - abs_b )/2;
  A9 = ( abs_b(iR, iD) - b(iR, iD) + abs_b - b )/4;

  %% Cálculo del avance de la ecuación
  du = zeros(size(u));
  for canal = 1:N3
  du(:,:,canal) = dt *...
     ( A1.*u(iL, iU, canal) + A2.*u(: , iU, canal) + A3.*u(iR,
        iU, canal) + ...
       A4.*u(iL, :, canal)              + A6.*u(iR, :, canal) +
          ...
       A7.*u(iL, iD, canal) + A8.*u(: , iD, canal) + A9.*u(iR, iD
          , canal) );
  end

  %% Realizar la difusión de la imagen
  for canal = 1:N3
     u(:, :, canal) = ( u(:, :, canal) + du(:, :, canal) ) ./ (
        1 - dt*A5 );
  end

end
```

Listing C.12: **demo.m**

```
clear all;
close all;

imageSelection = 3;

if imageSelection == 1

   [I, map] = imread('montyPyton.gif');

   I = ind2rgb(I, map);
   I = double(I);

   sizeI = size(I);
   N = normrnd(0, .1, sizeI);
   IN = I + N;
   time_step = .5;
   k = .005;

elseif imageSelection == 2

   [I, map] = imread('kids.tif');
```

```
    I = ind2rgb(I, map);
    IN = double(I);

    sizeI = size(I);
    time_step = .5;
    k = .005;

elseif imageSelection == 3

    [I, map] = imread('Libros.jpg');
    I = double(I)./255;
    IN = I;

    sizeI = size(I);

    time_step = .5;
    k = .005;

elseif imageSelection == 4
    [I, map] = imread('trianguloNoisy.bmp');

    sizeI = size(I);

    IN = double(I);

    time_step = .5;
    k = .1;
elseif imageSelection == 5
    [I, map] = imread('phantomNoisySigma25.bmp');

    sizeI = size(I);

    IN = double(I);

    time_step = .5;
    k = .1;
end


maxIters = 8;

t_init = tic();
image = anisotropicDiffusion(IN, maxIters, time_step, k, 0);
t_end = toc(t_init)

%% Muestra de los resultados
figure(101);
if (sizeI(1) >= sizeI(2))
    imshow([IN, image], []);
else
    imshow([IN; image], []);

end
title('I vs IN vs Restored');
```

C.2. Inpainting

Listing C.13: **dist.m**

```
function [ d ] = dist( p, q )
%DIST Calcula el peso de la distancia entre los puntos p y q
% Entradas:
%  · p — Componentes x,y del punto P
%  · q — Componentes x,y del punto Q
% Salidas:
%  · d — Factor de peso de la distancia entre las dos componentes

   d = ( p(1) - q(1) )^2 + ( p(2) - q(2) )^2;
   d = 1./sqrt(d);

end
```

Listing C.14: **aproxInpaint.m**

```
function [ image ] = aproxInpaint( image, mask, debugMode )
%APROXINPAINT Calcula la aproximación rápida a la solución del
   método de
%inpainting
% Entradas:
%  · image — Imagen dañada a donde se quiere aproximar el
   resultado de la
%  restauración
%  · mask — Máscara binaria que indica los puntos a restaurar
%  · debugMode — Modo de depuración con muestra de lso resultados
    a cada
%  paso del algoritmo
% Salidas:
%  · image — Imagen restaurada


   if ( max(image(:)) <= 1)
      image = image.*255;
      reescalar_imagen = 1;
   else
      reescalar_imagen = 0;
   end

   [height, width, channels] = size(image);

   mask = double(mask);

   imageOut = image;

   maskPoints = sum( mask(:) );
   maskEdge = laplacian( mask ) < 0;

   while ( maskPoints > 0 )

      for i = 2:height-1
```

```
        for j = 2:width-1
            if ( maskEdge(i,j) == 1)
                kernel = zeros(3);
                kernel(1,1) = ( 1 - mask(i-1,j-1) ) .* dist([i,j], [i-1, j-1]);
                kernel(1,2) = ( 1 - mask(i-1,j ) ) .* dist([i,j], [i-1, j ]);
                kernel(1,3) = ( 1 - mask(i-1,j+1) ) .* dist([i,j], [i-1, j+1]);
                kernel(2,1) = ( 1 - mask(i ,j-1) ) .* dist([i,j], [i , j-1]);
                kernel(2,3) = ( 1 - mask(i ,j+1) ) .* dist([i,j], [i , j+1]);
                kernel(3,1) = ( 1 - mask(i+1,j-1) ) .* dist([i,j], [i+1, j-1]);
                kernel(3,2) = ( 1 - mask(i+1,j ) ) .* dist([i,j], [i+1, j ]);
                kernel(3,3) = ( 1 - mask(i+1,j+1) ) .* dist([i,j], [i+1, j+1]);

                sumKernel = sum(kernel(:));
                for c = 1:channels
                   auxValue = image(i - 1, j - 1, c) * kernel(1, 1) ...
                                + image(i - 1, j , c) * kernel(1, 2) ...
                                + image(i - 1, j + 1, c) * kernel(1, 3) ...
                                + image( i, j - 1, c) * kernel(2, 1) ...
                                + image(i , j + 1, c) * kernel(2, 3) ...
                                + image(i + 1, j - 1, c) * kernel(3, 1) ...
                                + image(i + 1, j , c) * kernel(3, 2) ...
                                + image(i + 1, j + 1, c) * kernel(3, 3);
                   imageOut(i,j,c) = auxValue./sumKernel;

                end
            end
        end
    end

    % Show progress
    if ( debugMode == 1 )
        figure(30);
        hold off;
        imshow([image, imageOut]./255, []);
%         hold on;
%         contour([mask, mask], [0 0], 'b', 'LineWidth', 1);
%         drawnow;
%         pause;
```

```
      end

      image = imageOut;
      mask = mask - maskEdge;
      maskPoints = sum( mask(:) );
      maskEdge = laplacian( mask ) < 0;
   end


   if (reescalar_imagen == 1)
      image = image./255;
   end

end
```

Listing C.15: **inpaint.m**

```
function [ image ] = inpaint( image, mask, debugMode )
%INPAINT Realiza la restauración de la zona dañada indicada por
    la máscara
% Entradas:
%  · image - Imagen dañada a restaurar
%  · mask - Máscara binaria que indica los puntos a restaurar
%  · debugMode - Modo de depuración con muestra de lso resultados
     a cada
%  paso del algoritmo
% Salidas:
%  · image - Imagen restaurada

   if ( max(image(:)) <= 1)
      image = 255.*image;
      rescale_image = 1;
   else
      rescale_image = 0;
   end

   A = 15;
   B = 2;

   endFlag = 0;

   nIter = 1;
   if ( size(image,3) == 3 )
      maxIters = 30;
   else
      maxIters = 20;
   end

   I_original = image;

   % Se calcula la aproximación rápida a la restauración
   image = aproxInpaint( image, mask, 0 );

   if (size(image,3) == 3)
      image = rgb2pol(image);
```

```
   end

   while ( (~endFlag) && (nIter <= maxIters) )
   %% Bucle donde se realiza el inpainting
      for idxInpainting=1:A
         [image, endFlag] = InpaintIter( image, mask, .1 );
         if (endFlag)
            disp('endFlag es 1, se finaliza el proceso de
               inpainting');
            return;
         end
      end

   %% Bucle donde se realiza la difusión
      for idxDiffusion=1:B
         image = AnisotropicDiffusion( image, mask, .2 );
      end

      if ( (debugMode) && (mod(nIter, 1) == 0) )
         if ( size(image,3) == 3 )
            image_show = real( pol2rgb(image) );
         else
            image_show = image;
         end
         image_show( image_show < 0 ) = 0;
         image_show( image_show > 255 ) = 255;
         figure(30);
         subplot(121);
         imshow(I_original./255, []);
         subplot(122);
         imshow(image_show./255, []);
      end
      disp(['Iteración ', num2str(nIter), '/', num2str(maxIters)
          ]);
      nIter = nIter + 1;
%       endFlag = 0;
   end

   if ( size(image,3) == 3)
      image = pol2rgb(image);
      image = clip(image);
   end

   if (rescale_image == 1)
      image = image./255;
   end

   disp(['Numero final iteraciones ', num2str(nIter - 1)]);
end
```

Listing C.16: **InpaintIter.m**

```
function [ image, endFlag ] = InpaintIter( image, mask, dT )
%INPAINTITER Realiza una iteración del algoritmo de inpainting
% Entrada:
```

```
%  · image — Imagen que se va a restaurar
%  · mask — Máscara binaria que indica los píxeles que se deben
   restaurar
%  · dT — Variación temporal a utilizar
% Salida:
%  · image — Imagen resultado tras la iteración
%  · endFlag — Un flag que indica si la convergencia ya se ha
   alcanzado

   endFlag = 1;

   [height, width, channels] = size(image);

   L = laplacian(image);

   iL = [1, 1:height-1];
   iR = [2:height, height];
   iD = [1, 1:width-1]';
   iU = [2:width, width]';

   for c = 1:channels

      channel = image(:, :, c);

%       Ix = ( channel(iR, :) - channel(iL, :) )./2;
%       Iy = ( channel(:, iU) - channel(:, iD) )./2;
%
%       normD = sqrt(Ix.^2 + Iy.^2 + 1e-6 );
%       Nx = -Iy./normD;
%       Ny = Ix./normD;

      Nx = -( channel(:, iU) - channel(:, iD) )./2;
      Ny = ( channel(iR, :) - channel(iL, :) )./2;

      Ixb = channel - channel(iL, :);
      Ixbm = min(Ixb, 0);
      IxbM = max(Ixb, 0);
      Ixf = channel(iR, :) - channel;
      Ixfm = min(Ixf, 0);
      IxfM = max(Ixf, 0);
      Iyb = channel - channel(:, iD);
      Iybm = min(Iyb, 0);
      IybM = max(Iyb, 0);
      Iyf = channel(:, iU) - channel;
      Iyfm = min(Iyf, 0);
      IyfM = max(Iyf, 0);

      dLx = ( L(iR, :, c) - L(iL, :, c))./2;
      dLy = ( L(:, iU, c) - L(:, iD, c))./2;

      beta = (dLx.*Nx + dLy.*Ny);

      if ( max( abs(beta(:)) ) > 1 )
         endFlag = 0;
```

```
        end

        betaOverZero = beta > 0;
        betaUnderZero = beta < 0;

        normGradI = zeros([height, width]);

        normGradI(betaOverZero) = ...
            Ixbm(betaOverZero).^2 + IxfM(betaOverZero).^2 ...
            + Iybm(betaOverZero).^2 + IyfM(betaOverZero).^2;

        normGradI(betaUnderZero) = ...
            IxbM(betaUnderZero).^2 + Ixfm(betaUnderZero).^2 ...
            + IybM(betaUnderZero).^2 + Iyfm(betaUnderZero).^2;

        normGradI = sqrt(sqrt(normGradI));

        beta = beta.*normGradI;
        signBeta = sign(beta);

        I_t = mask.*signBeta.*sqrt( sqrt(signBeta.*beta) );

        image(:,:,c) = channel + dT.*I_t;

    end

end
```

Listing C.17: **laplacian.m**

```
function [ laplacian ] = laplacian( imagen )
%LAPLACIAN Calcula el laplaciano de la imagen seleccionada
% Entradas:
%  · image — Imagen de la que calcular el laplaciano
% Salidas:
%  · laplacian — Laplaciano de la imagen

    [height, width, channels] = size(imagen);

    iL = [1, 1:height-1];
    iR = [2:height, height];
    iD = [1, 1:width-1]';
    iU = [2:width, width]';

    laplacian = zeros([height, width, channels]);

    for c = 1:channels
        channel = imagen(:,:,c);
        laplacian(:,:,c) = channel(iL, :) + channel(iR,:) ...
            + channel(:, iD) + channel(:, iU) - 4*channel;
    end

end
```

Listing C.18: **AnisotropicDiffusion.m**

```
function [ image ] = AnisotropicDiffusion( image, mask, dT )
%ANISOTROPICDIFFUSION Filtrado de difusión anisótropa en la región indicada
%por la máscara
% Entradas:
%  · image — Imagen a filtrar
%  · mask — Máscara binaria indicando los píxeles donde se va a aplicar la
%  difusión
%  · dT — Variación temporal del esquema numérico a aplicar
% Salidas:
%  · image — Imagen filtrada

   [height, width, channels] = size(image);

   iL = [1, 1:height-1];
   iR = [2:height, height];
   iD = [1, 1:width-1]';
   iU = [2:width, width]';

   for c = 1:channels

      channel = image(:, :, c);

      dIx = ( channel(iR, :) - channel(iL, :) )./2;
      dIy = ( channel(:, iU) - channel(:, iD) )./2;

      dIxx = channel(iR, :) - 2.*channel + channel(iL, :);
      dIxy = ( channel(iR, iU) - channel(iR, iD) - channel(iL, iU) + channel(iL, iD) )./4;
      dIyy = channel(:, iU) - 2.*channel + channel(:, iD);

      normSquared = dIx.^2 + dIy.^2 + 1e-10;

      update = ( dIxx.*dIy.^2 - 2.*dIxy.*dIx.*dIy + dIyy.*dIx.^2 )./normSquared;

      image(:,:,c) = image(:,:,c) + mask.*dT.*update;

   end

end
```

Listing C.19: **rgb2pol.m**

```
function [ image ] = rgb2pol( imageRGB )
%RGB2POL Conversión de una imagen en formato RGB a un formato representado
%por el módulo de las componentes y sus coordenadas polares
% Entradas:
%  · imageRGB — Imagen RGB de la que se va a realizar la conversión
% Salidas:
```

```
%  · image — Imagen en coordenadas polares

   sizeImage = size(imageRGB);
   image = zeros(sizeImage);

   R = imageRGB(:,:,1);
   G = imageRGB(:,:,2);
   B = imageRGB(:,:,3);
   rho = sqrt( R.^2 + G.^2 + B.^2 );
   image(:,:,1) = rho;
   image(:,:,2) = G./rho;
   image(:,:,3) = R./sqrt( R.^2 + B.^2 );

end
```

Listing C.20: **pol2rgb.m**

```
function [ imageRGB ] = pol2rgb( image )
%POL2RGB Conversión de una imagen en formato representado por el
    módulo de
%las componentes y sus coordenadas polares a formato RGB
% Entradas:
%  · image — Imagen en coordenadas porlares de la que se va a
   realizar la
%  conversión
% Salidas:
%  · imageRGB — Imagen en formato RGB

   sizeImage = size(image);
   imageRGB = zeros(sizeImage);

   rho = image(:,:,1);
   sinPsi = image(:,:,2);
   sinTheta = image(:,:,3);

% Green
   G = rho .* sinPsi;
% Red
   R = sqrt( rho.^2 - G.^2 ) .* sinTheta;
% Blue
   B = sqrt( rho.^2 - R.^2 - G.^2 );

   imageRGB(:,:,1) = R;
   imageRGB(:,:,2) = G;
   imageRGB(:,:,3) = B;

end
```

Listing C.21: **clip.m**

```
function [ image ] = clip( image )
%CLIP Elimina valores indeseados y los redondea para poder
   volver a
%almacenar la imagen en un formato estándar
% Entradas:
```

```
% · image — Imagen a retocar
% Salidas:
% · image — Imagen retocada

   image = round(real(image));

   image(image < 0) = 0;
   image(image > 255) = 255;

end
```

Listing C.22: **demo.m**

```
clear all;
close all;

selectedImage = 4;

if (selectedImage == 1)

   addpath('.\Synthetic\');

   [I, map] = imread('.\Synthetic\Numbers.png');
   I = double( rgb2gray(I) );
   mask = (I == 84);

elseif (selectedImage == 2)

   addpath('.\Synthetic\');

   [I, map] = imread('.\Synthetic\Square.png');
   I = double( rgb2gray(I) );
   mask = (I == 129);

elseif (selectedImage == 3)

   addpath('.\Synthetic\');

   [I, map] = imread('.\Synthetic\Circle.png');
   I = double( rgb2gray(I) );
   mask = (I == 129);

elseif (selectedImage == 4)

   addpath('.\Niñas\');

   I = double( imread('Original.gif') );
   M = double( imread('Masked.gif') );

   mask = zeros(size(M));
   mask(M == 212) = 1;

elseif (selectedImage == 5)

   I = double( imread('LennaHoles.gif') );
```

```
   mask = zeros(size(I));
   mask(I == 255) = 1;

elseif (selectedImage == 6)

   [I, map] = imread('NewOrleans.gif');
   I = double( ind2rgb(I, map) );

   mask = zeros(size(I,1), size(I,2));
   mask( I(:,:,1) >= 0.98) = 1;

elseif (selectedImage == 7)

   [I, map] = imread('MontyPhyton.gif');
   I = double( ind2rgb(I, map) )*100;
   I = round(I)/100;

   maskR = zeros(size(I,1), size(I,2));
   maskG = zeros(size(I,1), size(I,2));
   maskB = zeros(size(I,1), size(I,2));
   maskR(I(:,:,1) >= 0.97) = 1;
   maskG(I(:,:,2) >= 0.97) = 1;
   maskB(I(:,:,3) >= 0.97) = 1;
   mask = maskR.*maskG.*maskB;

   h = ones(3)/9;
   mask = imfilter(mask, h, 'same', 'conv', 'replicate') > 0;

end

debugMode = 0;

t_init = tic();
   I_inpainted = inpaint(I, mask, debugMode);
t_end = toc(t_init);

figure;
imshow([I, I_inpainted], []);
hold on;
contour([mask, mask], [0 0], 'r', 'LineWidth', 1);
```

Apéndice D

Aplicación Android

D.1. Código C

Listing D.1: **util.h**

```
1  /*
2   * Modifica los valores para que se encuentren dentro del rango permitido por RGB
3   */
4  int clip(int v);
5  /*
6   * Devuelve el alpha correspodiente al píxel
7   */
8  int alpha(int c);
9  /*
10  * Devuelve el rojo correspondiente al píxel
11  */
12 int red(int c);
13 /*
14  * Devuelve el verde correspondiente al píxel
15  */
16 int green(int c);
17 /*
18  * Devuelve el azul correspondiente al píxel
19  */
20 int blue(int c);
21 /*
22  * Devuelve el valor compuesto RGB correspondiente al píxel tal como lo gestiona Android
23  */
24 int rgb(int r, int g, int b);
25 /*
26  * Calcula el índice correspondientes al punto x,y dentro del vector
27  */
28 int imageIndex(int x, int y);
29 /*
30  * Devuelve el mínimo entre los valores a y b
31  */
32 int min(float a, float b);
33 /*
34  * Devuelve el máximo entre los valores a y b
```

```
 */
int max(float a, float b);
/*
 * Devuelve -1 ne caso de que el valor sea negativo y 1 en caso de que sea positivo o cero
 */
float sign(float num);

/*
 * Verifica si la imagen de entrada es gris
 */
int isImageGray();
/*
 * Crea una versión en escala de grises de la imagen
 */
void setGrayScale();
```

Listing D.2: **util.c**

```
extern int bitmapWidth;
extern int bitmapHeight;
extern int bitmapChannelSize;
extern int bitmapChannels;
extern float *grayScaleImage;
extern float *image;

int clip(int v) {
	return (v < 255 ? (v < 0 ? 0 : v) : 255);
}
int alpha(int c) {
	return (c >> 24) & 0xff;
}
int red(int c) {
	return (c >> 16) & 0xff;
}
int green(int c) {
	return (c >> 8) & 0xff;
}
int blue(int c) {
	return (c & 0xff);
}
int rgb(int r, int g, int b) {
	return (0xff000000 | (((r & 0xff) << 16) | ((g & 0xff) << 8) | (b & 0xff)));
}
int imageIndex(int x, int y) {

	if (x < 0) {
		x = 0;
	} else if (x >= bitmapWidth) {
		x = bitmapWidth - 1;
	}
	if (y < 0) {
		y = 0;
	} else if (y >= bitmapHeight) {
		y = bitmapHeight - 1;
	}
```

```

	return y * bitmapWidth + x;
}

float sign(float num) {
	if (num < 0) {
		return -1;
	} else {
		return 1;
	}

}

int isImageGray() {
	int i;
	for (i = 0; i < bitmapChannelSize; i++) {
		if ((image[i] != image[i + bitmapChannelSize])
				|| (image[i] != image[i + 2 * bitmapChannelSize])) {
			return 0;
		}
	}
	return 1;
}
void setGrayScale() {
	if (bitmapChannels == 1) {
		grayScaleImage = image;
	} else {
		int i;
		for (i = 0; i < bitmapChannelSize; i++) {
			grayScaleImage[i] = 0.2989 * image[i]
					+ 0.58705 * image[i + bitmapChannelSize]
					+ 0.11405 * image[i + 2 * bitmapChannelSize];
		}
	}
}
```

Listing D.3: **anisotropicDiffusion.c**

```
#include <string.h>
#include <jni.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <math.h>
#include <util.h>
#include <android/log.h>
#include <android/bitmap.h>

#define LOG_TAG "anisotropic_diffusion_C"
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO,LOG_TAG,__VA_ARGS__)
#define LOGE(...) __android_log_print(ANDROID_LOG_ERROR,LOG_TAG,__VA_ARGS__)
typedef struct {
	uint16_t x;
	uint16_t y;
} point;
```

```
18 struct structureTensor {
19       float Jxx;
20       float Jxy;
21       float Jyy;
22 };
23 struct eigens {
24       float mu1;
25       float vx;
26       float vy;
27       float mu2;
28       float ux;
29       float uy;
30 };
31 struct diffusionTensor {
32       float Dxx;
33       float Dxy;
34       float Dyy;
35 };
36
37 struct structureTensor *calcStructureTensor();
38 struct eigens *calcEigenValuesAndVectors(struct structureTensor *structTensor);
39 struct diffusionTensor *calcDiffusionTensor(
40             struct eigens *eigenValuesAndVectors, float k);
41 void diffusionScheme(struct diffusionTensor *diffusionTensors, float deltaT);
42
43 int bitmapWidth;
44 int bitmapHeight;
45 int bitmapChannelSize;
46 int bitmapChannels;
47 float *grayScaleImage;
48 float *image;
49 /*
50  * Calculates the estructure tensor for all the image and returns it
51  */
52 struct structureTensor *calcStructureTensor() {
53       struct structureTensor *sT = (struct structureTensor *) malloc(
54                   bitmapChannelSize * sizeof(struct structureTensor));
55
56       LOGI("calcStructureTensor() BEGIN");
57
58       LOGI("Gaussian Filter Creation");
59
60       float filter[5][5] = { { 0.0030, 0.0133, 0.0219, 0.0133, 0.0030 }, { 0.0133,
61                   0.0596, 0.0983, 0.0596, 0.0133 }, { 0.0219, 0.0983, 0.1621, 0.0983,
62                   0.0219 }, { 0.0133, 0.0596, 0.0983, 0.0596, 0.0133 }, { 0.0030,
63                   0.0133, 0.0219, 0.0133, 0.0030 } };
64
65       int x, y, i, j, idx;
66       float value, *imageFGaussian;
67       imageFGaussian = (float *) malloc(bitmapChannelSize * sizeof(float));
68
69       LOGI("Applying Gaussian Filter");
70
71       // Apply gaussian filter 2D
```

```
72      int idx1, idx2, idx3, idx4, idx5, idx6, idx7, idx8, idx9, idx10, idx11,
73                  idx12, idx13, idx14, idx15, idx16, idx17, idx18, idx19, idx20,
74                  idx21, idx22, idx23, idx24, idx25;
75      for (x = 0; x < bitmapWidth; x++) {
76            for (y = 0; y < bitmapHeight; y++) {
77                  idx1 = imageIndex(x - 2, y - 2);
78                  idx2 = imageIndex(x - 1, y - 2);
79                  idx3 = imageIndex(x, y - 2);
80                  idx4 = imageIndex(x + 1, y - 2);
81                  idx5 = imageIndex(x + 2, y - 2);
82                  idx6 = imageIndex(x - 2, y - 1);
83                  idx7 = imageIndex(x - 1, y - 1);
84                  idx8 = imageIndex(x, y - 1);
85                  idx9 = imageIndex(x + 1, y - 1);
86                  idx10 = imageIndex(x + 2, y - 1);
87                  idx11 = imageIndex(x - 2, y);
88                  idx12 = imageIndex(x - 1, y);
89                  idx13 = imageIndex(x, y);
90                  idx14 = imageIndex(x + 1, y);
91                  idx15 = imageIndex(x + 2, y);
92                  idx16 = imageIndex(x - 2, y + 1);
93                  idx17 = imageIndex(x - 1, y + 1);
94                  idx18 = imageIndex(x, y + 1);
95                  idx19 = imageIndex(x + 1, y + 1);
96                  idx20 = imageIndex(x + 2, y + 1);
97                  idx21 = imageIndex(x - 2, y + 2);
98                  idx22 = imageIndex(x - 1, y + 2);
99                  idx23 = imageIndex(x, y + 2);
100                 idx24 = imageIndex(x + 1, y + 2);
101                 idx25 = imageIndex(x + 2, y + 2);
102
103                 imageFGaussian[idx13] = filter[0][0] * grayScaleImage[idx1]
104                             + filter[0][1] * grayScaleImage[idx2]
105                             + filter[0][2] * grayScaleImage[idx3]
106                             + filter[0][3] * grayScaleImage[idx4]
107                             + filter[0][4] * grayScaleImage[idx5]
108                             + filter[1][0] * grayScaleImage[idx6]
109                             + filter[1][1] * grayScaleImage[idx7]
110                             + filter[1][2] * grayScaleImage[idx8]
111                             + filter[1][3] * grayScaleImage[idx9]
112                             + filter[1][4] * grayScaleImage[idx10]
113                             + filter[2][0] * grayScaleImage[idx11]
114                             + filter[2][1] * grayScaleImage[idx12]
115                             + filter[2][2] * grayScaleImage[idx13]
116                             + filter[2][3] * grayScaleImage[idx14]
117                             + filter[2][4] * grayScaleImage[idx15]
118                             + filter[3][0] * grayScaleImage[idx16]
119                             + filter[3][1] * grayScaleImage[idx17]
120                             + filter[3][2] * grayScaleImage[idx18]
121                             + filter[3][3] * grayScaleImage[idx19]
122                             + filter[3][4] * grayScaleImage[idx20]
123                             + filter[4][0] * grayScaleImage[idx21]
124                             + filter[4][1] * grayScaleImage[idx22]
125                             + filter[4][2] * grayScaleImage[idx23]
```

```
                              + filter[4][3] * grayScaleImage[idx24]
                              + filter[4][4] * grayScaleImage[idx25];

            }
      }

      LOGI("Calculus of derivatives");

      // Calc dirivatives respect X and Y (Dx y Dy)
      float derivativeX[3][3] = { { 0.09375, 0, -0.09375 },
                  { 0.3125, 0, -0.3125 }, { 0.09375, 0, -0.09375 } };
      float derivativeY[3][3] = { { 0.09375, 0.3125, 0.09375 }, { 0, 0, 0 }, {
                  -0.09375, -0.3125, -0.09375 } };

      float *Dx = (float *) malloc(bitmapChannelSize * sizeof(float));
      float *Dy = (float *) malloc(bitmapChannelSize * sizeof(float));
      for (x = 0; x < bitmapWidth; x++) {
            for (y = 0; y < bitmapHeight; y++) {
                  idx1 = imageIndex(x - 1, y - 1);
                  idx2 = imageIndex(x, y - 1);
                  idx3 = imageIndex(x + 1, y - 1);
                  idx4 = imageIndex(x - 1, y);
                  idx5 = imageIndex(x, y);
                  idx6 = imageIndex(x + 1, y);
                  idx7 = imageIndex(x - 1, y + 1);
                  idx8 = imageIndex(x, y + 1);
                  idx9 = imageIndex(x + 1, y + 1);

                  Dx[idx5] = derivativeX[0][0] * imageFGaussian[idx1]
                              + derivativeX[1][0] * imageFGaussian[idx2]
                              + derivativeX[2][0] * imageFGaussian[idx3]
                              + derivativeX[0][2] * imageFGaussian[idx7]
                              + derivativeX[1][2] * imageFGaussian[idx8]
                              + derivativeX[2][2] * imageFGaussian[idx9];

                  Dy[idx5] = derivativeY[0][0] * imageFGaussian[idx1]
                              + derivativeY[0][1] * imageFGaussian[idx4]
                              + derivativeY[0][2] * imageFGaussian[idx7]
                              + derivativeY[2][0] * imageFGaussian[idx3]
                              + derivativeY[2][1] * imageFGaussian[idx6]
                              + derivativeY[2][2] * imageFGaussian[idx9];

            }
      }

      free(imageFGaussian);

      LOGI("Calculus of Tensor Product");

      struct structureTensor *auxST = (struct structureTensor *) malloc(
                  bitmapChannelSize * sizeof(struct structureTensor));
      // Execution of the tensor product
      /*for (i = 0; i < bitmapChannelSize; i++) {
       sT[i].Jxx = Dx[i] * Dx[i];
```

```
180         sT[i].Jxy = Dx[i] * Dy[i];
181         sT[i].Jyy = Dy[i] * Dy[i];
182         }
183 
184         free(Dx);
185         free(Dy);*/
186 
187       for (i = 0; i < bitmapChannelSize; i++) {
188             auxST[i].Jxx = Dx[i] * Dx[i];
189             auxST[i].Jxy = Dx[i] * Dy[i];
190             auxST[i].Jyy = Dy[i] * Dy[i];
191       }
192 
193       free(Dx);
194       free(Dy);
195 
196       LOGI("Applying Gaussian Filter 2nd time");
197 
198       // Aplicar filtro gaussiano a los resultados obtenidos
199       for (x = 0; x < bitmapWidth; x++) {
200             for (y = 0; y < bitmapHeight; y++) {
201                   idx1 = imageIndex(x - 2, y - 2);
202                   idx2 = imageIndex(x - 1, y - 2);
203                   idx3 = imageIndex(x, y - 2);
204                   idx4 = imageIndex(x + 1, y - 2);
205                   idx5 = imageIndex(x + 2, y - 2);
206                   idx6 = imageIndex(x - 2, y - 1);
207                   idx7 = imageIndex(x - 1, y - 1);
208                   idx8 = imageIndex(x, y - 1);
209                   idx9 = imageIndex(x + 1, y - 1);
210                   idx10 = imageIndex(x + 2, y - 1);
211                   idx11 = imageIndex(x - 2, y);
212                   idx12 = imageIndex(x - 1, y);
213                   idx13 = imageIndex(x, y);
214                   idx14 = imageIndex(x + 1, y);
215                   idx15 = imageIndex(x + 2, y);
216                   idx16 = imageIndex(x - 2, y + 1);
217                   idx17 = imageIndex(x - 1, y + 1);
218                   idx18 = imageIndex(x, y + 1);
219                   idx19 = imageIndex(x + 1, y + 1);
220                   idx20 = imageIndex(x + 2, y + 1);
221                   idx21 = imageIndex(x - 2, y + 2);
222                   idx22 = imageIndex(x - 1, y + 2);
223                   idx23 = imageIndex(x, y + 2);
224                   idx24 = imageIndex(x + 1, y + 2);
225                   idx25 = imageIndex(x + 2, y + 2);
226 
227                   sT[idx13].Jxx = filter[0][0] * auxST[idx1].Jxx
228                               + filter[0][1] * auxST[idx2].Jxx
229                               + filter[0][2] * auxST[idx3].Jxx
230                               + filter[0][3] * auxST[idx4].Jxx
231                               + filter[0][4] * auxST[idx5].Jxx
232                               + filter[1][0] * auxST[idx6].Jxx
233                               + filter[1][1] * auxST[idx7].Jxx
```

```
                            + filter[1][2] * auxST[idx8].Jxx
                            + filter[1][3] * auxST[idx9].Jxx
                            + filter[1][4] * auxST[idx10].Jxx
                            + filter[2][0] * auxST[idx11].Jxx
                            + filter[2][1] * auxST[idx12].Jxx
                            + filter[2][2] * auxST[idx13].Jxx
                            + filter[2][3] * auxST[idx14].Jxx
                            + filter[2][4] * auxST[idx15].Jxx
                            + filter[3][0] * auxST[idx16].Jxx
                            + filter[3][1] * auxST[idx17].Jxx
                            + filter[3][2] * auxST[idx18].Jxx
                            + filter[3][3] * auxST[idx19].Jxx
                            + filter[3][4] * auxST[idx20].Jxx
                            + filter[4][0] * auxST[idx21].Jxx
                            + filter[4][1] * auxST[idx22].Jxx
                            + filter[4][2] * auxST[idx23].Jxx
                            + filter[4][3] * auxST[idx24].Jxx
                            + filter[4][4] * auxST[idx25].Jxx;

                sT[idx13].Jxy = filter[0][0] * auxST[idx1].Jxy
                            + filter[0][1] * auxST[idx2].Jxy
                            + filter[0][2] * auxST[idx3].Jxy
                            + filter[0][3] * auxST[idx4].Jxy
                            + filter[0][4] * auxST[idx5].Jxy
                            + filter[1][0] * auxST[idx6].Jxy
                            + filter[1][1] * auxST[idx7].Jxy
                            + filter[1][2] * auxST[idx8].Jxy
                            + filter[1][3] * auxST[idx9].Jxy
                            + filter[1][4] * auxST[idx10].Jxy
                            + filter[2][0] * auxST[idx11].Jxy
                            + filter[2][1] * auxST[idx12].Jxy
                            + filter[2][2] * auxST[idx13].Jxy
                            + filter[2][3] * auxST[idx14].Jxy
                            + filter[2][4] * auxST[idx15].Jxy
                            + filter[3][0] * auxST[idx16].Jxy
                            + filter[3][1] * auxST[idx17].Jxy
                            + filter[3][2] * auxST[idx18].Jxy
                            + filter[3][3] * auxST[idx19].Jxy
                            + filter[3][4] * auxST[idx20].Jxy
                            + filter[4][0] * auxST[idx21].Jxy
                            + filter[4][1] * auxST[idx22].Jxy
                            + filter[4][2] * auxST[idx23].Jxy
                            + filter[4][3] * auxST[idx24].Jxy
                            + filter[4][4] * auxST[idx25].Jxy;

                sT[idx13].Jyy = filter[0][0] * auxST[idx1].Jyy
                            + filter[0][1] * auxST[idx2].Jyy
                            + filter[0][2] * auxST[idx3].Jyy
                            + filter[0][3] * auxST[idx4].Jyy
                            + filter[0][4] * auxST[idx5].Jyy
                            + filter[1][0] * auxST[idx6].Jyy
                            + filter[1][1] * auxST[idx7].Jyy
                            + filter[1][2] * auxST[idx8].Jyy
                            + filter[1][3] * auxST[idx9].Jyy
```

```
                              + filter[1][4] * auxST[idx10].Jyy
                              + filter[2][0] * auxST[idx11].Jyy
                              + filter[2][1] * auxST[idx12].Jyy
                              + filter[2][2] * auxST[idx13].Jyy
                              + filter[2][3] * auxST[idx14].Jyy
                              + filter[2][4] * auxST[idx15].Jyy
                              + filter[3][0] * auxST[idx16].Jyy
                              + filter[3][1] * auxST[idx17].Jyy
                              + filter[3][2] * auxST[idx18].Jyy
                              + filter[3][3] * auxST[idx19].Jyy
                              + filter[3][4] * auxST[idx20].Jyy
                              + filter[4][0] * auxST[idx21].Jyy
                              + filter[4][1] * auxST[idx22].Jyy
                              + filter[4][2] * auxST[idx23].Jyy
                              + filter[4][3] * auxST[idx24].Jyy
                              + filter[4][4] * auxST[idx25].Jyy;

            }
      }

// Clean temporal variables
      free(auxST);

      LOGI("calcStructureTensor() END");

      return sT;
}
/*
 * Calculates the eigenvaules and eigenvectors of a structure tensor
 * and returns all of them
 */
struct eigens *calcEigenValuesAndVectors(struct structureTensor *structTensor) {

      LOGI("calcEigenValuesAndVectors() BEGIN");

      struct eigens *e = (struct eigens *) malloc(
                  bitmapChannelSize * sizeof(struct eigens));
      int i;
      float diffJxxJyy;
      float disc;
      float norm2;
      for (i = 0; i < bitmapChannelSize; i++) {
            // Calc discriminant
            diffJxxJyy = structTensor[i].Jxx - structTensor[i].Jyy;
            disc = sqrtf(powf(diffJxxJyy, 2) + 4 * powf(structTensor[i].Jxy, 2));

            // Assuming ux, calc uy
            e[i].ux = 2 * structTensor[i].Jxy;
            e[i].uy = structTensor[i].Jyy - structTensor[i].Jxx + disc;

            // Normalization of eigenvectors
            norm2 = sqrtf(powf(e[i].ux, 2) + powf(e[i].uy, 2));
            if (abs(norm2) > 10e-10) {
                  e[i].ux = e[i].ux / norm2;
```

```
342                   e[i].uy = e[i].uy / norm2;
343             }
344
345             // v and u are orthogonal
346             e[i].vx = -e[i].uy;
347             e[i].vy = e[i].ux;
348
349             // Calc of eigenvalues
350             e[i].mu1 = (structTensor[i].Jxx + structTensor[i].Jyy + disc) / 2;
351             e[i].mu2 = (structTensor[i].Jxx + structTensor[i].Jyy - disc) / 2;
352
353       }
354       LOGI("calcEigenValuesAndVectors() END");
355
356       return e;
357 }
358 /*
359  * Calculates the diffusion tensor of the image using the eigenvalues,
360  * the eigenvectors and a diffusion coeficient contributed by the user
361  */
362 struct diffusionTensor *calcDiffusionTensor(
363             struct eigens *eigenValuesAndVectors, float k) {
364
365       LOGI("calcDiffusionTensor() BEGIN");
366
367       struct diffusionTensor *dT = (struct diffusionTensor *) malloc(
368                   bitmapChannelSize * sizeof(struct diffusionTensor));
369       float c = 3.31488;
370       float m = 4;
371       float mu1;
372       float maxMu1 = 0;
373       float denom;
374       float exponential;
375       float *lambda1 = (float *) malloc(bitmapChannelSize * sizeof(float));
376       float *lambda2 = (float *) malloc(bitmapChannelSize * sizeof(float));
377
378       LOGI("Defining max fo mu1");
379
380       int i;
381       for (i = 0; i < bitmapChannelSize; i++) {
382             if (eigenValuesAndVectors[i].mu1 > maxMu1) {
383                   maxMu1 = eigenValuesAndVectors[i].mu1;
384             }
385       }
386
387       LOGI("Defining matrices lambda1 and lambda2");
388
389       for (i = 0; i < bitmapChannelSize; i++) {
390             mu1 = eigenValuesAndVectors[i].mu1;
391
392             if (mu1 <= 0)
393                   lambda2[i] = 1;
394             else {
395                   denom = powf((mu1 / maxMu1) / k, m);
```

```
                  exponential = expf(-c / denom);
                  lambda2[i] = 1 - exponential;
            }

            lambda1[i] = 1;

      }

      LOGI("Applying Diffusion Tensor");
      // Applied anisotropic filter defining Diffusion Tensor
      for (i = 0; i < bitmapChannelSize; i++) {
            dT[i].Dxx = lambda1[i] * powf(eigenValuesAndVectors[i].ux, 2)
                        + lambda2[i] * powf(eigenValuesAndVectors[i].vx, 2);
            dT[i].Dxy = lambda1[i] * eigenValuesAndVectors[i].ux
                        * eigenValuesAndVectors[i].uy
                        + lambda2[i] * eigenValuesAndVectors[i].vx
                                    * eigenValuesAndVectors[i].vy;
            dT[i].Dyy = lambda1[i] * powf(eigenValuesAndVectors[i].uy, 2)
                        + lambda2[i] * powf(eigenValuesAndVectors[i].vy, 2);
      }

      free(lambda1);
      free(lambda2);

      LOGI("calcDiffusionTensor() END");

      return dT;
}
/*
 * Performs the diffusion scheme advancing the result of the equation
 * an deltaT time step contributed by the user
 */
void diffusionScheme(struct diffusionTensor *diffusionTensors, float deltaT) {
      // Performs anisotropic diffusion semi-implicit scheme
      LOGI("diffusionScheme() BEGIN");
      int channel;
      int i;
      int j;

      float a1, a2, a3, a4, a6, a7, a8, a9;
      float *absDxy = (float *) malloc(bitmapChannelSize * sizeof(float));
      float *a5 = (float *) malloc(bitmapChannelSize * sizeof(float));
      float *du = (float *) malloc(
                  bitmapChannelSize * bitmapChannels * sizeof(float));

      for (i = 0; i < bitmapChannelSize; i++) {
            absDxy[i] = fabsf(diffusionTensors[i].Dxy);
      }

      LOGI("Calculus of Net values and Image Update");

      int idx1, idx2, idx3, idx4, idx5, idx6, idx7, idx8, idx9;
      for (i = 0; i < bitmapWidth; i++) {
            for (j = 0; j < bitmapHeight; j++) {
```

```
450                 // Calc indexes of net
451                 idx1 = imageIndex(i - 1, j - 1);
452                 idx2 = imageIndex(i, j - 1);
453                 idx3 = imageIndex(i + 1, j - 1);
454                 idx4 = imageIndex(i - 1, j);
455                 idx5 = imageIndex(i, j);
456                 idx6 = imageIndex(i + 1, j);
457                 idx7 = imageIndex(i - 1, j + 1);
458                 idx8 = imageIndex(i, j + 1);
459                 idx9 = imageIndex(i + 1, j + 1);
460
461                 // Calc of net values
462                 a1 = (absDxy[idx1] - diffusionTensors[idx1].Dxy + absDxy[idx5]
463                             - diffusionTensors[idx5].Dxy) / 4;
464
465                 a2 = (diffusionTensors[idx2].Dyy + diffusionTensors[idx5].Dyy
466                             - absDxy[idx2] - absDxy[idx5]) / 2;
467
468                 a3 = (absDxy[idx3] + diffusionTensors[idx3].Dxy + absDxy[idx5]
469                             + diffusionTensors[idx5].Dxy) / 4;
470
471                 a4 = (diffusionTensors[idx4].Dxx + diffusionTensors[idx5].Dxx
472                             - absDxy[idx4] - absDxy[idx5]) / 2;
473
474                 a5[idx5] = (-(diffusionTensors[idx4].Dxx
475                             + 2 * diffusionTensors[idx5].Dxx
476                             + diffusionTensors[idx6].Dxx)
477                             + (absDxy[idx2] + absDxy[idx4] + 2 * absDxy[idx5]
478                                         + absDxy[idx6] + absDxy[idx8])
479                             - (diffusionTensors[idx2].Dyy
480                                         + 2 * diffusionTensors[idx5].Dyy
481                                         + diffusionTensors[idx8].Dyy)) / 2
482                             - (absDxy[idx1] - diffusionTensors[idx1].Dxy + absDxy[idx3]
483                                         + diffusionTensors[idx3].Dxy + absDxy[idx7]
484                                         + diffusionTensors[idx7].Dxy + absDxy[idx9]
485                                         - diffusionTensors[idx9].Dxy) / 4;
486
487                 a6 = (diffusionTensors[idx6].Dxx + diffusionTensors[idx5].Dxx
488                             - absDxy[idx6] - absDxy[idx5]) / 2;
489
490                 a7 = (absDxy[idx7] + diffusionTensors[idx7].Dxy + absDxy[idx5]
491                             + diffusionTensors[idx5].Dxy) / 4;
492
493                 a8 = (diffusionTensors[idx8].Dyy + diffusionTensors[idx5].Dyy
494                             - absDxy[idx8] - absDxy[idx5]) / 2;
495
496                 a9 = (absDxy[idx9] - diffusionTensors[idx9].Dxy + absDxy[idx5]
497                             - diffusionTensors[idx5].Dxy) / 4;
498
499                 // Image update calculus
500                 du[idx5] = deltaT
501                             * (a1 * image[idx1] + a2 * image[idx2] + a3 * image[idx3]
502                                         + a4 * image[idx4] + a6 * image[idx6]
503                                         + a7 * image[idx7] + a8 * image[idx8]
```

```
                                        + a9 * image[idx9]);
                  if (bitmapChannels == 3) {
                        du[idx5 + bitmapChannelSize] = deltaT
                                    * (a1 * image[idx1 + bitmapChannelSize]
                                                + a2 * image[idx2 + bitmapChannelSize]
                                                + a3 * image[idx3 + bitmapChannelSize]
                                                + a4 * image[idx4 + bitmapChannelSize]
                                                + a6 * image[idx6 + bitmapChannelSize]
                                                + a7 * image[idx7 + bitmapChannelSize]
                                                + a8 * image[idx8 + bitmapChannelSize]
                                                + a9 * image[idx9 + bitmapChannelSize]);
                        du[idx5 + 2 * bitmapChannelSize] = deltaT
                                    * (a1 * image[idx1 + 2 * bitmapChannelSize]
                                                + a2 * image[idx2 + 2 * bitmapChannelSize]
                                                + a3 * image[idx3 + 2 * bitmapChannelSize]
                                                + a4 * image[idx4 + 2 * bitmapChannelSize]
                                                + a6 * image[idx6 + 2 * bitmapChannelSize]
                                                + a7 * image[idx7 + 2 * bitmapChannelSize]
                                                + a8 * image[idx8 + 2 * bitmapChannelSize]
                                                + a9 * image[idx9 + 2 * bitmapChannelSize]);
                  }

            }
      }

      LOGI("Image update");
      // Image update
      for (i = 0; i < bitmapChannelSize * bitmapChannels; i++) {
            image[i] = (image[i] + du[i])
                        / (1 - deltaT * a5[i  % bitmapChannelSize]);
      }

      free(a5);
      free(du);
      free(absDxy);

      LOGI("diffusionScheme() END");
}

JNIEXPORT void JNICALL
     Java_com_imageRestoration_anisotropicDiffusion_AnisotropicDiffusionActivity_anisotropicDiffusion
     (
            JNIEnv * env, jobject obj, jintArray inputPixels, jint width,
            jint height, jint maxIters, jdouble timeStep, jdouble k,
            jintArray outputPixels) {

      bitmapWidth = width;
      bitmapHeight = height;
      bitmapChannelSize = bitmapWidth * bitmapHeight;
      bitmapChannels = 3;

      LOGI(
                  "IMAGE INFO: Width %i - Height %i - Channels %i - ChannelSize %i",
                       bitmapWidth, bitmapHeight, bitmapChannels, bitmapChannelSize);
```

```
555
556       jint *sourcePixels;
557       jint *resultPixels;
558
559       sourcePixels = (*env)->GetIntArrayElements(env, inputPixels, 0);
560       resultPixels = (*env)->GetIntArrayElements(env, outputPixels, 0);
561
562       // Declaración de la imagen con la que se va a trabajar
563       image = (float *) malloc(
564                   bitmapChannelSize * bitmapChannels * sizeof(float));
565       grayScaleImage = (float *) malloc(bitmapChannelSize * sizeof(float));
566
567       // Asignación de la imagen de entrada a image para trabajar con ella
568       int color, idx;
569
570       for (idx = 0; idx < bitmapChannelSize; idx++) {
571             color = sourcePixels[idx];
572             image[idx] = (float) red(color);
573             image[idx + bitmapChannelSize] = (float) green(color);
574             image[idx + 2 * bitmapChannelSize] = (float) blue(color);
575       }
576
577       if (isImageGray() == 1) {
578             bitmapChannels = 1;
579             LOGI("The image is originally in GRAY scale");
580       } else {
581             LOGI("The image is originally in RGB color scale");
582       }
583       setGrayScale();
584
585       LOGI("Anisotropic Diffusion BEGIN");
586
587       int iter;
588       struct structureTensor *structTensor;
589       struct eigens *eigenValuesAndVectors;
590       struct diffusionTensor *diffusionTensors;
591
592       for (iter = 1; iter <= maxIters; iter++) {
593             // Si la imagen es en color pasarla a escala de grises
594             if (bitmapChannels == 3) {
595                   setGrayScale();
596             }
597
598             // Calcular tensor de estructura
599             LOGI(
600                         "Calcs of Estructure Tensors at iteration %i of %i", iter, maxIters);
601             structTensor = calcStructureTensor();
602             // Calcular autovalores y autovectores del tensor de estructura de cada uno de los
                  puntos de la imagen
603             LOGI(
604                         "Calcs of eigenvalues and eigenvectors at iteration %i of %i", iter,
                              maxIters);
605             eigenValuesAndVectors = calcEigenValuesAndVectors(structTensor);
606             free(structTensor);
```

```
607             // Calcular tensor de difusión
608             LOGI(
609                         "Calcs of Diffusion Tensors at iteration %i of %i with k=%f", iter,
                                maxIters, k);
610             diffusionTensors = calcDiffusionTensor(eigenValuesAndVectors,
611                         (float) k);
612             free(eigenValuesAndVectors);
613             // Realizar actualización de la imagen
614             LOGI(
615                         "Diffusion Scheme at iteration %i of %i con timeStep=%f", iter,
                                maxIters, timeStep);
616             diffusionScheme(diffusionTensors, (float) timeStep);
617             free(diffusionTensors);
618       }
619
620       LOGI("Anisotropic Diffusion END");
621
622       LOGI("unlocking pixels");
623
624       // Asignación de la variable image a la imagen de salida
625       int value, r, g, b;
626       for (idx = 0; idx < bitmapChannelSize; idx++) {
627             if (bitmapChannels == 3) {
628                   r = clip((int) roundf(image[idx]));
629                   g = clip((int) roundf(image[idx + bitmapChannelSize]));
630                   b = clip((int) roundf(image[idx + bitmapChannelSize * 2]));
631                   resultPixels[idx] = (jint) rgb(r, g, b);
632             } else {
633                   value = clip((int) roundf(image[idx]));
634                   resultPixels[idx] = (jint) rgb(value, value, value);
635             }
636       }
637
638       LOGI("Cleaning variables");
639
640       free(image);
641       if (bitmapChannels != 1) {
642             free(grayScaleImage);
643       }
644
645       LOGI("Assigning outputs");
646
647       (*env)->ReleaseIntArrayElements(env, outputPixels, resultPixels, 0);
648
649 }
```

Listing D.4: **imageInpainting.c**

```
1  #include <string.h>
2  #include <jni.h>
3  #include <stdio.h>
4  #include <stdint.h>
5  #include <stdlib.h>
6  #include <math.h>
7  #include <util.h>
```

```
8  #include <android/log.h>
9  #include <android/bitmap.h>
10
11 #define LOG_TAG "anisotropic_diffusion_C"
12 #define LOGI(...) __android_log_print(ANDROID_LOG_INFO,LOG_TAG,__VA_ARGS__)
13 #define LOGE(...) __android_log_print(ANDROID_LOG_ERROR,LOG_TAG,__VA_ARGS__)
14 struct point {
15       int x;
16       int y;
17 };
18
19 float maskLaplacian(int x, int y);
20 void updateBinaryMask(int maskEdgePixels, struct point *maskEdge);
21 void approximateSolution();
22 float *laplacian();
23 int inpaintIter(float deltaT);
24 void anisotropicDiffusion(float deltaT, int useMask);
25 void imageInpainting();
26
27 int bitmapWidth;
28 int bitmapHeight;
29 int bitmapChannelSize;
30 int bitmapChannels;
31 float *image;
32 struct point *mask;
33 int maskPixels;
34 short int *binaryMask;
35 int dataEdgePixels;
36 struct point *dataEdge;
37
38 /**
39  * Computes the laplacian value of the binary mask in the point x, y
40  */
41 float maskLaplacian(int x, int y) {
42
43       int iN, iS, iW, iE, iC;
44       float laplacian = 0;
45
46       iN = imageIndex(x, y - 1);
47       iS = imageIndex(x, y + 1);
48       iW = imageIndex(x - 1, y);
49       iE = imageIndex(x + 1, y);
50       iC = imageIndex(x, y);
51
52       laplacian = binaryMask[iN] + binaryMask[iS] + binaryMask[iE]
53                   + binaryMask[iW] - 4 * binaryMask[iC];
54
55       return laplacian;
56
57 }
58
59 /**
60  * Updates the binary mask setting to zero the points on maskEdge
61  */
```

```
void updateBinaryMask(int maskEdgePixels, struct point *maskEdge) {

      int i, x, y, idx;

      for (i = 0; i < maskEdgePixels; i++) {
            x = maskEdge[i].x;
            y = maskEdge[i].y;
            idx = imageIndex(x, y);

            binaryMask[idx] = 0;
      }

}

/**
 * Performs an approximation to the solution of the inpainting algorithm
 */
void approximateSolution() {

      int i, j, c, offset, x, y, idx1, idx2, idx3, idx4, idx5, idx6, idx7, idx8,
                  idx9;

      float k1, k2, k3, k4, k6, k7, k8, k9, sumaK;

      int maskEdgePixels;
      struct point *maskEdge;

      // Extraer píxeles del borde de la máscara
      maskEdgePixels = 0;
      for (i = 0; i < bitmapWidth; i++) {
            for (j = 0; j < bitmapHeight; j++) {
                  if (maskLaplacian(i, j) < 0) {
                        maskEdgePixels++;
                  }
            }
      }

      if (maskEdgePixels > 0) {
            maskEdge = (struct point *) malloc(
                        maskEdgePixels * sizeof(struct point));

            c = 0;
            for (i = 0; i < bitmapWidth; i++) {
                  for (j = 0; j < bitmapHeight; j++) {
                        if (maskLaplacian(i, j) < 0) {
                              maskEdge[c].x = i;
                              maskEdge[c].y = j;
                              c++;
                        }
                  }
            }
      }

      while (maskEdgePixels > 0) {
```

```
116
117             // Aproximación de los píxeles de la frontera
118             for (i = 0; i < maskEdgePixels; i++) {
119                   x = maskEdge[i].x;
120                   y = maskEdge[i].y;
121
122                   idx1 = imageIndex(x - 1, y - 1);
123                   idx2 = imageIndex(x, y - 1);
124                   idx3 = imageIndex(x + 1, y - 1);
125                   idx4 = imageIndex(x - 1, y);
126                   idx5 = imageIndex(x, y);
127                   idx6 = imageIndex(x + 1, y);
128                   idx7 = imageIndex(x - 1, y + 1);
129                   idx8 = imageIndex(x, y + 1);
130                   idx9 = imageIndex(x + 1, y + 1);
131
132                   k1 = (1 - binaryMask[idx1]) / sqrt(2);
133                   k2 = (1 - binaryMask[idx2]);
134                   k3 = (1 - binaryMask[idx3]) / sqrt(2);
135                   k4 = (1 - binaryMask[idx4]);
136                   k6 = (1 - binaryMask[idx6]);
137                   k7 = (1 - binaryMask[idx7]) / sqrt(2);
138                   k8 = (1 - binaryMask[idx8]);
139                   k9 = (1 - binaryMask[idx9]) / sqrt(2);
140
141                   sumaK = k1 + k2 + k3 + k4 + k6 + k7 + k8 + k9;
142
143                   for (c = 0; c < bitmapChannels; c++) {
144
145                         offset = c * bitmapChannelSize;
146
147                         image[idx5 + offset] = (k1 * image[idx1 + offset]
148                                     + k2 * image[idx2 + offset] + k3 * image[idx3 + offset]
149                                     + k4 * image[idx4 + offset] + k6 * image[idx6 + offset]
150                                     + k7 * image[idx7 + offset] + k8 * image[idx8 + offset]
151                                     + k9 * image[idx9 + offset]) / sumaK;
152
153                   }
154
155             }
156
157             // Actualización de la máscara binaria
158             updateBinaryMask(maskEdgePixels, maskEdge);
159
160             // Extraer píxeles del borde de la máscara
161             maskEdgePixels = 0;
162             for (i = 0; i < bitmapWidth; i++) {
163                   for (j = 0; j < bitmapHeight; j++) {
164                         if (maskLaplacian(i, j) < 0) {
165                               maskEdgePixels++;
166                         }
167                   }
168             }
169
```

```
170             if (maskEdgePixels > 0) {
171                   maskEdge = (struct point *) malloc(
172                               maskEdgePixels * sizeof(struct point));
173
174                   c = 0;
175                   for (i = 0; i < bitmapWidth; i++) {
176                         for (j = 0; j < bitmapHeight; j++) {
177                               if (maskLaplacian(i, j) < 0) {
178                                     maskEdge[c].x = i;
179                                     maskEdge[c].y = j;
180                                     c++;
181                               }
182                         }
183                   }
184             }
185
186       }
187
188 }
189
190 /*
191  * Calculates and returns the discrete laplacian of the image
192  */
193 float *laplacian() {
194
195       //LOGI("Laplacian calculus BEGIN");
196
197       int c, i, j, x, y, iN, iS, iE, iW, iC, offset;
198
199       float *laplacian = malloc(
200                   bitmapChannels * bitmapChannelSize * sizeof(float));
201
202       for (c = 0; c < bitmapChannels; c++) {
203
204             offset = c * bitmapChannelSize;
205
206             /*  for (i = 1; i < bitmapWidth - 1; i++) {
207              for (j = 1; j < bitmapHeight - 1; j++) {
208              iN = imageIndex(i, j - 1) + offset;
209              iS = imageIndex(i, j + 1) + offset;
210              iW = imageIndex(i - 1, j) + offset;
211              iE = imageIndex(i + 1, j) + offset;
212              iC = imageIndex(i, j) + offset;
213
214              laplacian[iC] = image[iN] + image[iS] + image[iE] + image[iW]
215              - 4 * image[iC];
216              }
217              }*/
218
219             for (i = 0; i < maskPixels; i++) {
220                   x = mask[i].x;
221                   y = mask[i].y;
222
223                   iN = imageIndex(x, y - 1) + offset;
```

```
                  iS = imageIndex(x, y + 1) + offset;
                  iW = imageIndex(x - 1, y) + offset;
                  iE = imageIndex(x + 1, y) + offset;
                  iC = imageIndex(x, y) + offset;

                  laplacian[iC] = image[iN] + image[iS] + image[iE] + image[iW]
                              - 4 * image[iC];
            }

            for (j = 0; j < dataEdgePixels; j++) {
                  x = dataEdge[j].x;
                  y = dataEdge[j].y;

                  iN = imageIndex(x, y - 1) + offset;
                  iS = imageIndex(x, y + 1) + offset;
                  iW = imageIndex(x - 1, y) + offset;
                  iE = imageIndex(x + 1, y) + offset;
                  iC = imageIndex(x, y) + offset;

                  laplacian[iC] = image[iN] + image[iS] + image[iE] + image[iW]
                              - 4 * image[iC];
            }
      }

      //LOGI("Laplacian calculus END");

      return laplacian;

}

/*
 * Performs an iteration of the inpainting algorithm with a time step deltaT
 */
int inpaintIter(float deltaT) {

      int i, c, x, y, offset, endFlag, idxUp, idxDown, idxLeft, idxRight,
                  idxCenter;
      float dIx, dIy, normDI, Nx, Ny, dLx, dLy, beta, normGradI, Ixb, Ixf, Iyb,
                  Iyf, signBeta;
      float *L, *update;

      endFlag = 1;

      L = laplacian();

      //LOGI("Inpaint Iteration BEGIN");

      update = (float *) malloc(bitmapChannels * maskPixels * sizeof(float));

      for (c = 0; c < bitmapChannels; c++) {

            offset = c * bitmapChannelSize;

            for (i = 0; i < maskPixels; i++) {
```

```
                x = mask[i].x;
                y = mask[i].y;
                idxUp = imageIndex(x, y - 1) + offset;
                idxDown = imageIndex(x, y + 1) + offset;
                idxLeft = imageIndex(x - 1, y) + offset;
                idxRight = imageIndex(x + 1, y) + offset;
                idxCenter = imageIndex(x, y) + offset;

                Nx = -(image[idxDown] - image[idxUp]) / 2;
                Ny = (image[idxRight] - image[idxLeft]) / 2;

                dLx = (L[idxRight] - L[idxLeft]);
                dLy = (L[idxDown] - L[idxUp]);

                beta = (dLx * Nx + dLy * Ny) / 2;

                if (fabsf(beta) > 1) {
                      endFlag = 0;
                }

                Ixb = image[idxCenter] - image[idxLeft];
                Ixf = image[idxRight] - image[idxCenter];
                Iyb = image[idxCenter] - image[idxUp];
                Iyf = image[idxDown] - image[idxCenter];

                if (beta > 0) {
                      Ixb = fminf(Ixb, 0);
                      Ixf = fmaxf(Ixf, 0);
                      Iyb = fminf(Iyb, 0);
                      Iyf = fmaxf(Iyf, 0);
                } else {
                      Ixb = fmaxf(Ixb, 0);
                      Ixf = fminf(Ixf, 0);
                      Iyb = fmaxf(Iyb, 0);
                      Iyf = fminf(Iyf, 0);
                }

                normGradI = sqrtf(
                            powf(Ixb, 2) + powf(Ixf, 2) + powf(Iyb, 2) + powf(Iyf, 2));

                beta = beta * normGradI;

                signBeta = sign(beta);

                update[i + c * maskPixels] = deltaT * signBeta
                            * sqrtf(sqrtf(signBeta * beta));

          }

     }

     free(L);

     for (i = 0; i < maskPixels; i++) {
```

```
332             idxCenter = imageIndex(mask[i].x, mask[i].y);
333             image[idxCenter] = image[idxCenter] + update[i];
334             if (bitmapChannels == 3) {
335                   image[idxCenter + bitmapChannelSize] = image[idxCenter
336                               + bitmapChannelSize] + update[i + maskPixels];
337                   image[idxCenter + 2 * bitmapChannelSize] = image[idxCenter
338                               + 2 * bitmapChannelSize] + update[i + 2 * maskPixels];
339             }
340       }
341
342       free(update);
343
344       //LOGI("Inpaint Iteration END");
345
346       return endFlag;
347 }
348
349 /*
350  * Performs an iteration of the anisotropic diffusion algotithm
351  * for inpainting
352  */
353 void anisotropicDiffusion(float deltaT, int useMask) {
354
355       //LOGI("Anisotropic Diffusion BEGIN");
356
357       int i, j, c, offset, x, y, idx1, idx2, idx3, idx4, idx5, idx6, idx7, idx8,
358                   idx9;
359       float dIx, dIy, dIxx, dIxy, dIyy, normSquared;
360
361       if (useMask == 1) {
362
363             for (c = 0; c < bitmapChannels; c++) {
364
365                   offset = c * bitmapChannelSize;
366
367                   for (i = 0; i < maskPixels; i++) {
368                         x = mask[i].x;
369                         y = mask[i].y;
370                         idx1 = imageIndex(x - 1, y - 1) + offset;
371                         idx2 = imageIndex(x, y - 1) + offset;
372                         idx3 = imageIndex(x + 1, y - 1) + offset;
373                         idx4 = imageIndex(x - 1, y) + offset;
374                         idx5 = imageIndex(x, y) + offset;
375                         idx6 = imageIndex(x + 1, y) + offset;
376                         idx7 = imageIndex(x - 1, y + 1) + offset;
377                         idx8 = imageIndex(x, y + 1) + offset;
378                         idx9 = imageIndex(x + 1, y + 1) + offset;
379
380                         dIx = (image[idx6] - image[idx4]) / 2;
381                         dIy = (image[idx8] - image[idx2]) / 2;
382                         dIxx = (image[idx6] - 2 * image[idx5] + image[idx4]);
383                         dIxy = (image[idx9] - image[idx7] - image[idx3] + image[idx1])
384                                     / 4;
385                         dIyy = (image[idx8] - 2 * image[idx5] + image[idx2]);
```

```
386
387                         normSquared = dIx * dIx + dIy * dIy + 1e-10;
388                         image[idx5] = image[idx5]
389                                     + deltaT
390                                                 * (dIx * dIx * dIyy - 2 * dIx * dIy * dIxy
391                                                             + dIy * dIy * dIxx) / normSquared;
392
393                   }
394
395             }
396
397       } else {
398
399             for (c = 0; c < bitmapChannels; c++) {
400
401                   offset = c * bitmapChannelSize;
402
403                   for (i = 0; i < bitmapWidth; i++) {
404                         for (j = 0; j < bitmapHeight; j++) {
405                               idx1 = imageIndex(x - 1, y - 1) + offset;
406                               idx2 = imageIndex(x, y - 1) + offset;
407                               idx3 = imageIndex(x + 1, y - 1) + offset;
408                               idx4 = imageIndex(x - 1, y) + offset;
409                               idx5 = imageIndex(x, y) + offset;
410                               idx6 = imageIndex(x + 1, y) + offset;
411                               idx7 = imageIndex(x - 1, y + 1) + offset;
412                               idx8 = imageIndex(x, y + 1) + offset;
413                               idx9 = imageIndex(x + 1, y + 1) + offset;
414
415                               dIx = (image[idx6] - image[idx4]) / 2;
416                               dIy = (image[idx8] - image[idx2]) / 2;
417                               dIxx = (image[idx6] - 2 * image[idx5] + image[idx4]);
418                               dIxy = (image[idx9] - image[idx7] - image[idx3]
419                                           + image[idx1]) / 4;
420                               dIyy = (image[idx8] - 2 * image[idx5] + image[idx2]);
421
422                               normSquared = dIx * dIx + dIy * dIy + 1e-10;
423                               image[idx5] = image[idx5]
424                                           + deltaT
425                                                       * (dIx * dIx * dIyy - 2 * dIx * dIy *
                                                                 dIxy
426                                                                       + dIy * dIy * dIxx) /
                                                                             normSquared;
427
428                         }
429                   }
430
431             }
432
433       }
434
435       //LOGI("Anisotropic Diffusion END");
436
437 }
```

```
438
439 void imageInpainting() {
440
441       LOGI("Image Inpainting BEGIN");
442
443       int A = 15, B = 2;
444       int i, iter = 1, x, y, idx = 1;
445       int endFlag = 0;
446       int maxIters = 5;
447
448       if (bitmapChannels == 1) {
449             maxIters = 30;
450       } else if (bitmapChannels == 3) {
451             maxIters = 20;
452       }
453
454       while ((endFlag == 0) && (iter <= maxIters)) {
455             LOGI("Iteration %i of %i", iter, maxIters);
456             for (i = 0; i < A; i++) {
457                   endFlag = inpaintIter(0.1);
458                   if (endFlag == 1) {
459                         LOGI(
460                                     "The image update is under the prestablished threshold,
                                         inpainting process is stopped");
461                         return;
462                   }
463             }
464
465             for (i = 0; i < B; i++) {
466                   anisotropicDiffusion(0.2, 1);
467             }
468
469             iter++;
470       }
471
472       LOGI("Image Inpainting END");
473
474 }
475
476 JNIEXPORT void JNICALL
      Java_com_imageRestoration_inpainting_ImageInpaintingActivity_imageInpainting(
477             JNIEnv * env, jobject obj, jintArray inputPixels, jintArray inputMask,
478             jint width, jint height, jintArray outputPixels) {
479
480       bitmapWidth = width;
481       bitmapHeight = height;
482       bitmapChannelSize = bitmapWidth * bitmapHeight;
483       bitmapChannels = 3;
484
485       LOGI(
486                   "IMAGE INFO: Width %i - Height %i - Channels %i - ChannelSize %i",
                         bitmapWidth, bitmapHeight, bitmapChannels, bitmapChannelSize);
487
488       jint *sourcePixels;
```

```
489       jint *sourceMask;
490       jint *resultPixels;
491
492       sourcePixels = (*env)->GetIntArrayElements(env, inputPixels, 0);
493       sourceMask = (*env)->GetIntArrayElements(env, inputMask, 0);
494       resultPixels = (*env)->GetIntArrayElements(env, outputPixels, 0);
495
496       // Declaración de la imagen con la que se va a trabajar
497       image = (float *) malloc(
498                   bitmapChannelSize * bitmapChannels * sizeof(float));
499       //grayScaleImage = (float *) malloc(bitmapChannelSize * sizeof(float));
500       binaryMask = (short int *) malloc(bitmapChannelSize * sizeof(short int));
501
502       // Asignación de la imagen de entrada a image para trabajar con ella y recuento de los
              píxeles que están a uno en la máscara para poder reservar memoria para almacenar
              sus puntos
503       int i, j, color, idxMask, idxDataEdgePixels, idx;
504       float maskLaplacianValue;
505       maskPixels = 0;
506       dataEdgePixels = 0;
507
508       for (i = 0; i < bitmapChannelSize; i++) {
509             color = sourcePixels[i];
510             image[i] = (float) red(color);
511             image[i + bitmapChannelSize] = (float) green(color);
512             image[i + 2 * bitmapChannelSize] = (float) blue(color);
513             if (alpha(sourceMask[i]) != 0) {
514                   maskPixels++;
515                   binaryMask[i] = 1;
516             } else {
517                   binaryMask[i] = 0;
518             }
519       }
520
521       for (i = 0; i < bitmapWidth; i++) {
522             for (j = 0; j < bitmapHeight; j++) {
523                   maskLaplacianValue = maskLaplacian(i, j);
524                   if (maskLaplacianValue > 0) {
525                         dataEdgePixels++;
526                   }
527             }
528       }
529
530       LOGI("Mask Pixels %i", maskPixels);
531       mask = (struct point *) malloc(maskPixels * sizeof(struct point));
532       dataEdge = (struct point *) malloc(dataEdgePixels * sizeof(struct point));
533       // Creación de la máscara
534       idxMask = 0;
535       idxDataEdgePixels = 0;
536       for (i = 0; i < bitmapWidth; i++) {
537             for (j = 0; j < bitmapHeight; j++) {
538                   idx = imageIndex(i, j);
539                   if (binaryMask[idx] == 1) {
540                         mask[idxMask].x = i;
```

```
541                         mask[idxMask].y = j;
542                         idxMask++;
543                   } else {
544                         maskLaplacianValue = maskLaplacian(i, j);
545                         if (maskLaplacianValue > 0) {
546                               dataEdge[idxDataEdgePixels].x = i;
547                               dataEdge[idxDataEdgePixels].y = j;
548                               idxDataEdgePixels++;
549                         }
550                   }
551             }
552       }
553
554       if (isImageGray()) {
555             bitmapChannels = 1;
556             LOGI("The image is originally in gray scale");
557       } else {
558             bitmapChannels = 3;
559             LOGI("The image is originally in RGB");
560       }
561
562       approximateSolution();
563
564       LOGI("Approximation of the solution computed");
565
566       imageInpainting();
567
568       LOGI("unlocking pixels");
569
570       // Asignación de la variable image a la imagen de salida
571       int value, r, g, b;
572       for (i = 0; i < bitmapChannelSize; i++) {
573             if (bitmapChannels == 3) {
574                   r = clip((int) roundf(image[i]));
575                   g = clip((int) roundf(image[i + bitmapChannelSize]));
576                   b = clip((int) roundf(image[i + bitmapChannelSize * 2]));
577                   resultPixels[i] = (jint) rgb(r, g, b);
578             } else {
579                   value = clip((int) round(image[i]));
580                   resultPixels[i] = (jint) rgb(value, value, value);
581             }
582       }
583
584       LOGI("Cleaning variables");
585
586       free(image);
587       free(mask);
588       free(dataEdge);
589
590       LOGI("Assigning outputs");
591
592       (*env)->ReleaseIntArrayElements(env, outputPixels, resultPixels, 0);
593
594 }
```

Listing D.5: **Android.mk**

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE := PhotoRestore
### Add all source file names to be included in lib separated by a whitespace
LOCAL_SRC_FILES := anisotropicDiffusion.c imageInpainting.c util.c
LOCAL_LDLIBS := -llog -ljnigraphics

include $(BUILD_SHARED_LIBRARY)
```

D.2. Código Java

Listing D.6: **AnisotropicDiffusionActivity.java**

```
package com.imageRestoration.anisotropicDiffusion;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.res.Configuration;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
import android.widget.TextView;

import com.imageRestoration.R;
import com.imageRestoration.showImage.ShowImageActivity;
import com.imageRestoration.util.ImageFileManager;

/**
 * El objetivo de esta clase es implementar los métodos necesarios para realizar
 * el filtrado de difusión anisótropa
 *
 * @author Samuel Martinez
 */
public class AnisotropicDiffusionActivity extends Activity implements
            OnSeekBarChangeListener {

     /**
```

```
38        * Este bloque de código carga la librería nativa de C que contiene las
39        * funciones que se van a utilizar mediante JNI para mejorar el rendimiento
40        * del algoritmo
41        */
42       static {
43             // Carga de la librería nativa en C
44             String libName = "PhotoRestore";
45             System.loadLibrary(libName);
46       }
47
48       /**
49        * Firma de la función nativa para realizar el filtrado de difusión
50        * anisótropa
51        *
52        * @param pixelsInput
53        * Píxeles de entrada
54        * @param width
55        * Anchura de la imagen de entrada
56        * @param height
57        * Altura de la imagen de entrada
58        * @param maxIters
59        * Número máximo de iteraciones
60        * @param timeStep
61        * Tamaño del paso temporal
62        * @param k
63        * Coeficiente de difusividad
64        * @param pixelsOutput
65        * Píxeles de salida (donde se devolverá el valor de la imagen
66        * filtrada) mediante un paso de parámetros por referencia
67        */
68       private native void anisotropicDiffusion(int pixelsInput[], int width,
69                   int height, int maxIters, double timeStep, double k,
70                   int pixelsOutput[]);
71
72       public final static int ANISOTROPIC_DIFFUSION_PARAMS_DIALOG = 0;
73       public final static int ANISOTROPIC_DIFFUSION_HELP_DIALOG = 1;
74
75       private Bitmap bitmap;
76
77       private EditText numItersEditText;
78       private final static int minItersAllowed = 1;
79       private final static int maxItersAllowed = 100;
80       private int numIters;
81       private EditText numItersTextView;
82
83       private SeekBar timeStepSeekBar;
84       private final double minTimeStepAllowed = 0.01;
85       private final double maxTimeStepAllowed = 3;
86       private double timeStep;
87       private TextView timeStepTextView;
88
89       private SeekBar kSeekBar;
90       private final double minKAllowed = 0.001;
91       private final double maxKAllowed = 0.1;
```

```
92      private double k;
93      private TextView kTextView;
94
95      private TextView paramsErrorTextView;
96
97      /* GETTERS Y SETTERS */
98
99      private int getProgressK(double k) {
100           return Math
101                   .round((float) ((k - this.minKAllowed) * 100 / (this.maxKAllowed -
                          this.minKAllowed)));
102     }
103
104     private int getProgressTimeStep(double timeStep) {
105           return Math
106                   .round((float) ((timeStep - this.minTimeStepAllowed) * 100 / (this.
                          maxTimeStepAllowed - this.minTimeStepAllowed)));
107     }
108
109     private double getValueK(int progressK) {
110           double value = (this.minKAllowed + progressK
111                   * (this.maxKAllowed - this.minKAllowed) / 100);
112           return value;
113     }
114
115     private double getValueTimeStep(int progressTimeStep) {
116           return (this.minTimeStepAllowed + progressTimeStep
117                   * (this.maxTimeStepAllowed - this.minTimeStepAllowed) / 100);
118     }
119
120     @Override
121     protected void onCreate(Bundle savedInstanceState) {
122           super.onCreate(savedInstanceState);
123           setContentView(R.layout.anisotropic_diffusion);
124
125           Bundle bundle = getIntent().getExtras();
126           String imagePath = bundle.getString("IMAGE_PATH");
127
128           this.bitmap = ImageFileManager.loadBitmapARGB_8888(imagePath, true);
129
130           if (this.bitmap == null) {
131                 Log.e(AnisotropicDiffusionActivity.class.toString(), "La imagen "
132                             + imagePath + " no ha sido cargada");
133                 onCancelRestoration();
134           } else {
135
136                 ImageView imageView = (ImageView) findViewById(R.id.
                        anisotropic_diffusion_background);
137                 imageView.setImageBitmap(this.bitmap);
138
139                 // Valores por defecto de los parámetros para la difusión anisótropa
140                 this.numIters = 5;
141                 this.timeStep = .5;
142                 this.k = 0.005;
```

```
143
144             }
145
146       }
147
148       @Override
149       public void onConfigurationChanged(Configuration newConfig) {
150             super.onConfigurationChanged(newConfig);
151             setContentView(R.layout.anisotropic_diffusion);
152
153             if (this.bitmap == null) {
154                   Log.e(ShowImageActivity.class.toString() + ".onCreate()",
155                               "La imagen no ha sido cargada");
156                   finish();
157             } else {
158                   ImageView imageView = (ImageView) findViewById(R.id.
                          anisotropic_diffusion_background);
159                   imageView.setImageBitmap(this.bitmap);
160             }
161       }
162
163       /**
164        * Crea la ventana de diálogo y especifica las operaciones a realizar con
165        * ella
166        *
167        * @return Ventana de diálogo para introducción de los parámetros de
168        * difusión anisótropa
169        */
170       private Dialog createAnisotropicDiffusionParamsDialog() {
171             // Creación de las variable para albergar la ventana de diálogo
172             AlertDialog.Builder builder = new AlertDialog.Builder(this);
173             Dialog dialog;
174
175             // Toma de la configuración visual del XML que mapea la ventana de
176             // diálogo y creación de la disposición
177             LayoutInflater factory = LayoutInflater.from(this);
178             final View layout = factory.inflate(
179                         R.layout.anisotropic_diffusion_params_dialog, null);
180
181             // Estableciendo los campos textuales de los parámetros y escritura de
182             // sus valores por defecto escribir sus valores
183             this.numItersTextView = (EditText) layout
184                         .findViewById(R.id.anisotropic_diffusion_num_iters_value);
185             this.numItersTextView.setText(this.numIters + "");
186
187             this.timeStepTextView = (TextView) layout
188                         .findViewById(R.id.anisotropic_diffusion_time_step_value);
189             this.timeStepTextView.setText(this.timeStep + "");
190
191             this.kTextView = (TextView) layout
192                         .findViewById(R.id.anisotropic_diffusion_coefficient_value);
193             this.kTextView.setText(this.k + "");
194
195             this.paramsErrorTextView = (TextView) layout
```

```
196                         .findViewById(R.id.anisotropic_diffusion_params_error);
197
198             // Crear variables para obtener los valores introducidos por el usuario
199             numItersEditText = (EditText) layout
200                         .findViewById(R.id.anisotropic_diffusion_num_iters_value);
201             numItersEditText.setText(numIters + "");
202
203             kSeekBar = (SeekBar) layout
204                         .findViewById(R.id.anisotropic_diffusion_coefficient_adjust_bar);
205             kSeekBar.setOnSeekBarChangeListener(this);
206             kSeekBar.setProgress(getProgressK(this.k));
207
208             timeStepSeekBar = (SeekBar) layout
209                         .findViewById(R.id.anisotropic_diffusion_time_step_adjust_bar);
210             timeStepSeekBar.setOnSeekBarChangeListener(this);
211             timeStepSeekBar.setProgress(getProgressTimeStep(this.timeStep));
212
213             final AnisotropicDiffusionActivity activity = this;
214
215             // Creación del listener para el botón "Aceptar"
216             DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                    OnClickListener() {
217
218                   public void onClick(DialogInterface dialog, int which) {
219
220                         int numItersInput = Integer.parseInt(numItersEditText.getText()
221                                     .toString());
222                         k = getValueK(kSeekBar.getProgress());
223                         // Double.parseDouble(kTextView.getText().toString());
224                         timeStep = getValueTimeStep(timeStepSeekBar.getProgress());
225                         // Double.parseDouble(timeStepTextView.getText().toString());
226
227                         // Comprobación de que el número de iteraciones es correcto
228                         boolean params_error = false;
229                         if ((numItersInput < minItersAllowed)
230                                     || (numItersInput > maxItersAllowed)) {
231                               params_error = true;
232                               CharSequence currentErrorText = paramsErrorTextView
233                                           .getText();
234                               String error = getString(R.string.
                                          anisotropic_diffusion_error_param_num_iters);
235                               if (currentErrorText.length() == 0) {
236                                     paramsErrorTextView.setText(error);
237                               } else {
238                                     paramsErrorTextView.setText(currentErrorText + "\n"
239                                                 + error);
240                               }
241                         } else {
242                               numIters = numItersInput;
243                         }
244
245                         if (!params_error) {
246
247                               dialog.dismiss();
```

```
248
249                         final ProgressDialog progressDialog = ProgressDialog.show(
250                                     activity, "Difusión anisótropa",
251                                     "Por favor, espere");
252
253                         new Thread(new Runnable() {
254
255                               public void run() {
256                                     int width = bitmap.getWidth();
257                                     int height = bitmap.getHeight();
258                                     int pixelsInput[] = new int[width * height];
259                                     bitmap.getPixels(pixelsInput, 0, width, 0, 0,
260                                                 width, height);
261                                     int pixelsOutput[] = new int[width * height];
262
263                                     // Launch anisotropic diffusion process
264
265                                     final long t_init = System.currentTimeMillis();
266
267                                     anisotropicDiffusion(pixelsInput, width, height,
268                                                 numIters, timeStep, k, pixelsOutput);
269
270                                     final long t_end = System.currentTimeMillis();
271
272                                     bitmap = bitmap.copy(Bitmap.Config.ARGB_8888, true);
273                                     bitmap.setPixels(pixelsOutput, 0, width, 0, 0,
274                                                 width, height);
275
276                                     Log.i(AnisotropicDiffusionActivity.class.toString()
277                                                 + ".
                                                       createAnisotropicDiffusionParamsDialog
                                                       ()",
278                                                 "TIEMPO EMPLEADO EN LA DIFUSIÓN
                                                       ANISÓTROPA:"
279                                                       + (t_end - t_init) / 1000
280                                                       + "segundos");
281
282                                     progressDialog.dismiss();
283
284                                     // Se finaliza la ejecución del algoritmo enviándolo
285                                     // de vuelta al activity que lo lanzó
286                                     onRestorationSuccess();
287                               }
288                         }).start();
289
290                   }
291             }
292       };
293
294       DialogInterface.OnClickListener negativeButtonListener = new DialogInterface.
              OnClickListener() {
295
296             public void onClick(DialogInterface dialog, int which) {
297
```

```
298                         dialog.dismiss();
299
300                   }
301             };
302
303             // Establecimiento de los valores de la ventana de diálogo y creación
304             builder = new AlertDialog.Builder(this);
305             builder.setIcon(android.R.drawable.ic_menu_help);
306             builder.setTitle(getString(R.string.help));
307             builder.setView(layout);
308             builder.setPositiveButton(R.string.accept, positiveButtonListener);
309             builder.setNegativeButton(R.string.cancel, negativeButtonListener);
310             dialog = builder.create();
311
312             return dialog;
313       }
314
315       /**
316        * Crea la ventana de diálogo y especifica las operaciones a realizar con
317        * ella
318        *
319        * @return Ventana de diálogo con instrucciones de ayuda para el filtrado de
320        * difusión anisótropa
321        */
322       protected Dialog createHelpDialog() {
323
324             // Creación de las variable para albergar la ventana de diálogo
325             AlertDialog.Builder builder = new AlertDialog.Builder(this);
326             Dialog dialog;
327
328             // Toma de la configuración visual del XML que mapea la ventana de
329             // diálogo y creación de la disposición
330             LayoutInflater factory = LayoutInflater.from(this);
331             View layout = factory.inflate(
332                         R.layout.anisotropic_diffusion_help_dialog, null);
333
334             // Creación del listener para el botón "Aceptar"
335             DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                  OnClickListener() {
336
337                   public void onClick(DialogInterface dialog, int which) {
338
339                         dialog.dismiss();
340
341                   }
342             };
343
344             // Establecimiento de los valores de la ventana de diálogo y creación
345             builder = new AlertDialog.Builder(this);
346             builder.setIcon(android.R.drawable.ic_menu_help);
347             builder.setTitle(getString(R.string.help));
348             builder.setView(layout);
349             builder.setPositiveButton(R.string.accept, positiveButtonListener);
350             dialog = builder.create();
```

```
351
352             return dialog;
353       }
354
355       @Override
356       protected Dialog onCreateDialog(int id) {
357             Dialog dialog;
358
359             switch (id) {
360             case ANISOTROPIC_DIFFUSION_PARAMS_DIALOG:
361                   dialog = createAnisotropicDiffusionParamsDialog();
362                   break;
363             case ANISOTROPIC_DIFFUSION_HELP_DIALOG:
364                   dialog = createHelpDialog();
365                   break;
366             default:
367                   dialog = null;
368                   break;
369             }
370
371             return dialog;
372       }
373
374       @Override
375       public boolean onCreateOptionsMenu(Menu menu) {
376             MenuInflater menuInflater = getMenuInflater();
377             menuInflater.inflate(R.menu.anisotropic_diffusion_menu, menu);
378             return true;
379       }
380
381       @Override
382       public boolean onOptionsItemSelected(MenuItem item) {
383
384             switch (item.getItemId()) {
385             case R.id.anisotropic_diffusion_accept_opt:
386                   showDialog(ANISOTROPIC_DIFFUSION_PARAMS_DIALOG);
387                   return true;
388             case R.id.anisotropic_diffusion_help_opt:
389                   showDialog(ANISOTROPIC_DIFFUSION_HELP_DIALOG);
390                   return true;
391             case R.id.anisotropic_diffusion_back_opt:
392                   onCancelRestoration();
393                   return true;
394             default:
395                   return super.onOptionsItemSelected(item);
396             }
397       }
398
399       /**
400        * Envío de resultados cuando la restauración se ha realizado correctamente
401        */
402       public void onRestorationSuccess() {
403             // Guardado del bitmap en memoria
404             String filePath = ImageFileManager.baseFolderPath.concat("temp.png");
```

```
405             if (ImageFileManager.saveBitmap(bitmap, "temp",
406                         Bitmap.CompressFormat.PNG)) {
407                   Intent intent = new Intent();
408                   Bundle extras = new Bundle();
409                   extras.putString("IMAGE_PATH_RESULT", filePath);
410                   intent.putExtras(extras);
411
412                   setResult(RESULT_OK, intent);
413                   finish();
414             } else {
415                   onCancelRestoration();
416             }
417       }
418
419       /**
420        * Envío de resultados cuando se cancela el proceso de restauración
421        */
422       public void onCancelRestoration() {
423             setResult(Activity.RESULT_CANCELED);
424             finish();
425       }
426
427       @Override
428       public void onBackPressed() {
429
430             onCancelRestoration();
431       }
432
433       /**
434        * Modifica el progreso de las barras de elección de parámetros cuando son
435        * modificadas
436        */
437       public void onProgressChanged(SeekBar seekBar, int progress,
438                   boolean fromUser) {
439             String text = null;
440             if (seekBar.getId() == R.id.anisotropic_diffusion_time_step_adjust_bar) {
441                   text = String.format("%.2f", getValueTimeStep(progress));
442                   timeStepTextView.setText(text);
443             } else if (seekBar.getId() == R.id.anisotropic_diffusion_coefficient_adjust_bar) {
444                   text = String.format("%.4f", getValueK(progress));
445                   kTextView.setText(text);
446             }
447             Log.i(AnisotropicDiffusionActivity.class.toString()
448                         + ".onProgressChanged()", "Valor: " + text);
449       }
450
451       public void onStartTrackingTouch(SeekBar seekBar) {
452
453             // Auto-generated method stub
454
455       }
456
457       public void onStopTrackingTouch(SeekBar seekBar) {
458
```

```
459             // Auto-generated method stub
460
461         }
462
463 }
```

Listing D.7: **FileBrowserActivity.java**

```
1  package com.imageRestoration.fileBrowser;
2
3  import java.io.File;
4  import java.util.ArrayList;
5  import java.util.List;
6
7  import android.app.AlertDialog;
8  import android.app.Dialog;
9  import android.app.ListActivity;
10 import android.content.DialogInterface;
11 import android.content.Intent;
12 import android.graphics.Bitmap;
13 import android.graphics.BitmapFactory;
14 import android.os.Bundle;
15 import android.util.Log;
16 import android.view.LayoutInflater;
17 import android.view.Menu;
18 import android.view.MenuInflater;
19 import android.view.MenuItem;
20 import android.view.View;
21 import android.widget.ArrayAdapter;
22 import android.widget.ImageView;
23 import android.widget.ListView;
24 import android.widget.TextView;
25
26 import com.imageRestoration.R;
27 import com.imageRestoration.showImage.ShowImageActivity;
28
29 /**
30  * En esta clase se implementa el navegador de ficheros para la selección de la
31  * imagen a restaurar
32  *
33  * @author Samuel Martinez
34  *
35  */
36 public class FileBrowserActivity extends ListActivity {
37
38       private List<String> item = null;
39       private List<String> path = null;
40       private String root = "/";
41       private String initFolder = "/sdcard/";
42       private TextView myPath;
43       private File file;
44
45       // Constantes para la identificación de las ventanas de diálogo
46       private static final int FB_IMAGE_PREVIEW_DIALOG = 1;
47       private static final int FB_NOT_AN_IMAGE_DIALOG = 2;
```

```
48      private static final int FB_NOT_READABLE_FOLDER_DIALOG = 3;
49      private static final int FB_HELP_DIALOG = 4;
50
51      /** Called when the activity is first created. */
52      @Override
53      public void onCreate(Bundle savedInstanceState) {
54            super.onCreate(savedInstanceState);
55            setContentView(R.layout.explorer_main);
56
57            myPath = (TextView) findViewById(R.id.path);
58            getDir(initFolder);
59
60      }
61
62      /**
63       * Método donde, partir de dirPath, se muestra toda la estructura de
64       * ficheros residente en dentro de la carpeta
65       *
66       * @param dirPath
67       * Ruta a la carpeta
68       */
69      private void getDir(String dirPath) {
70
71            /*
72             * Cadena de texto auxiliar que almacena datos estándar que serán
73             * añadidos a la lista de navegación
74             */
75            String auxString = (String) getString(R.string.current_location) + " ";
76            myPath.setText(auxString + dirPath);
77
78            /*
79             * Creación de la estructura que almacenará la lista de
80             * ficheros/carpetas
81             */
82            item = new ArrayList<String>();
83            path = new ArrayList<String>();
84
85            File f = new File(dirPath);
86            File[] files = f.listFiles();
87
88            if (!dirPath.equals(initFolder)) {
89                  /*
90                   * En caso de que no se encuentre en la raíz del sistema de ficheros
91                   */
92                  auxString = (String) getString(R.string.sdcard_root);
93                  item.add(auxString);
94                  path.add(initFolder);
95
96            }
97
98            if (!dirPath.equals(root)) {
99                  /*
100                  * En caso de que no se encuentre en la raíz del sistema de ficheros
101                  */
```

```
102                 auxString = (String) getString(R.string.file_browser_back_folder);
103                 item.add(auxString);
104                 path.add(f.getParent());
105
106           }
107
108           /*
109            * Se recorre la lista que contiene los ficheros para añadir los
110            * elementos a una lista de items
111            */
112           for (int i = 0; i < files.length; i++) {
113                 File file = files[i];
114                 path.add(file.getPath());
115                 if (file.isDirectory()) {
116                       item.add(file.getName() + "/");
117                 } else {
118                       item.add(file.getName());
119                 }
120           }
121
122           ArrayAdapter<String> fileList = new ArrayAdapter<String>(this,
123                       R.layout.explorer_row, item);
124           setListAdapter(fileList);
125     }
126
127     /**
128      * Crea una ventana de diálogo que contiene una previsualización de la
129      * imagen que se desea cargar
130      *
131      * @return Ventana de diálogo
132      */
133     private Dialog createImagePreviewDialog() {
134           // Creación de las variable para albergar la ventana de diálogo
135           AlertDialog.Builder builder = new AlertDialog.Builder(this);
136           Dialog dialog;
137
138           // Toma de la configuración visual del XML que mapea la ventana de
139           // diálogo y creación de la disposición
140           LayoutInflater factory = LayoutInflater.from(this);
141           final View layout = factory.inflate(R.layout.image_load_dialog, null);
142
143           // Creación del listener para el botón "Aceptar"
144           DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                OnClickListener() {
145
146                 public void onClick(DialogInterface dialog, int which) {
147
148                       Intent intent = new Intent(FileBrowserActivity.this,
149                                   ShowImageActivity.class);
150
151                       Log.d(FileBrowserActivity.class.toString(), file
152                                   .getAbsoluteFile().toString());
153
154                       Bundle bundle = new Bundle();
```

```
155                         bundle.putString("IMAGE_PATH", file.getAbsolutePath()
156                                     .toString());
157                         intent.putExtras(bundle);
158
159                         startActivity(intent);
160
161                         dialog.dismiss();
162
163                   }
164             };
165
166             // Creación del listener para el botón "Cancelar"
167             DialogInterface.OnClickListener negativeButtonListener = new DialogInterface.
                    OnClickListener() {
168
169                   public void onClick(DialogInterface dialog, int which) {
170
171                         dialog.dismiss();
172
173                   }
174             };
175
176             // Establecimiento de los valores de la ventana de diálogo y creación
177             builder = new AlertDialog.Builder(this);
178             builder.setIcon(android.R.drawable.ic_menu_help);
179             builder.setTitle("[ ]");
180             builder.setView(layout);
181             builder.setPositiveButton(R.string.accept, positiveButtonListener);
182             builder.setNegativeButton(R.string.cancel, negativeButtonListener);
183             dialog = builder.create();
184
185             return dialog;
186       }
187
188       /**
189        * Crea una ventana de diálogo con el aviso de que el fichero no es una
190        * imagen
191        *
192        * @return Ventana de diálogo
193        */
194       private Dialog createNotAnImageDialog() {
195             // Creación de las variable para albergar la ventana de diálogo
196             AlertDialog.Builder builder = new AlertDialog.Builder(this);
197             Dialog dialog;
198
199             // Creación del listener para el botón "Aceptar"
200             DialogInterface.OnClickListener buttonListener = new DialogInterface.
                    OnClickListener() {
201
202                   public void onClick(DialogInterface dialog, int which) {
203
204                         dialog.dismiss();
205
206                   }
```

```
207             };
208
209             // Establecimiento de los valores de la ventana de diálogo y creación
210             builder.setIcon(android.R.drawable.ic_dialog_alert);
211             builder.setTitle("[]");
212             builder.setMessage(R.string.not_an_image_msg);
213             builder.setPositiveButton(R.string.accept, buttonListener);
214             dialog = builder.create();
215
216             return dialog;
217       }
218
219       /**
220        * Crea una ventana de diálogo donde se indica que la carpeta no es legible
221        *
222        * @return Ventana de diálogo
223        */
224       private Dialog createNotReadableFolderDialog() {
225             AlertDialog.Builder builder = new AlertDialog.Builder(this);
226             Dialog dialog;
227
228             DialogInterface.OnClickListener buttonListener = new DialogInterface.
                  OnClickListener() {
229
230                   public void onClick(DialogInterface dialog, int which) {
231                         dialog.dismiss();
232                   }
233             };
234
235             builder.setIcon(android.R.drawable.ic_dialog_info);
236             builder.setTitle("[]");
237             builder.setMessage(R.string.not_readable_folder_msg);
238             builder.setPositiveButton(R.string.accept, buttonListener);
239             dialog = builder.create();
240
241             return dialog;
242       }
243
244       /**
245        * Crea la ventana de diálogo y especifica las operaciones a realizar con
246        * ella
247        *
248        * @return Ventana de diálogo con instrucciones de ayuda
249        */
250       private Dialog createHelpDialog() {
251             // Creación de las variable para albergar la ventana de diálogo
252             AlertDialog.Builder builder = new AlertDialog.Builder(this);
253             Dialog dialog;
254
255             // Toma de la configuración visual del XML que mapea la ventana de
256             // diálogo y creación de la disposición
257             LayoutInflater factory = LayoutInflater.from(this);
258             final View layout = factory.inflate(R.layout.file_browser_help_dialog,
259                         null);
```

```
260
261             // Creación del listener para el botón "Cancelar"
262             DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                    OnClickListener() {
263
264                   public void onClick(DialogInterface dialog, int which) {
265
266                         dialog.dismiss();
267
268                   }
269             };
270
271             // Establecimiento de los valores de la ventana de diálogo y creación
272             builder = new AlertDialog.Builder(this);
273             builder.setIcon(android.R.drawable.ic_menu_help);
274             builder.setTitle(getString(R.string.help));
275             builder.setView(layout);
276             builder.setPositiveButton(R.string.accept, positiveButtonListener);
277             dialog = builder.create();
278
279             return dialog;
280       }
281
282       @Override
283       protected Dialog onCreateDialog(int id) {
284             Dialog dialogWindow;
285
286             switch (id) {
287             case FB_IMAGE_PREVIEW_DIALOG:
288                   dialogWindow = createImagePreviewDialog();
289                   break;
290             case FB_NOT_AN_IMAGE_DIALOG:
291                   dialogWindow = createNotAnImageDialog();
292                   break;
293             case FB_NOT_READABLE_FOLDER_DIALOG:
294                   dialogWindow = createNotReadableFolderDialog();
295                   break;
296             case FB_HELP_DIALOG:
297                   dialogWindow = createHelpDialog();
298                   break;
299             default:
300                   dialogWindow = null;
301                   break;
302             }
303
304             return dialogWindow;
305
306       }
307
308       /**
309        * Reconstruye la ventana de diálogo asignándole la nueva imagen
310        * seleccionada
311        *
312        * @param dialog
```

```
313         */
314         private void prepareImagePreviewDialog(Dialog dialog) {
315               // Se modifica el título de la venta de diálogo y se carga y muestra la
316               // imagen a previsualizar para solicitar al usuario confirmación para
317               // continuar al siguiente punto de la aplicación
318
319               if (file != null) {
320                     // Fijar el nuevo título de la ventana con el nombre del fichero
321                     // actual
322                     dialog.setTitle("[" + file.getName() + "]");
323
324                     // Extracción del recurso de la vista donde irá situada la imagen
325                     ImageView imageView = (ImageView) dialog
326                                 .findViewById(R.id.preview_image_dialog);
327                     // Carga de la imagen
328                     Bitmap bitmap = BitmapFactory.decodeFile(file.getAbsolutePath());
329                     // Inserción de la imagen en la vista
330                     imageView.setImageBitmap(bitmap);
331               } else {
332                     dialog.dismiss();
333               }
334         }
335
336         /**
337          * Reconstruye la ventana de diálogo asignándole el mensaje de que el
338          * fichero seleccionado no es una imagen
339          *
340          * @param dialog
341          */
342         private void prepareNotAnImageDialog(Dialog dialog) {
343               // Se modifica el título de la ventana de diálogo
344               dialog.setTitle("[" + file.getName() + "]");
345
346         }
347
348         /**
349          * Reconstruye la ventana de diálogo asignándole el mensaje de que el
350          * fichero o carpeta seleccionado no es legible
351          *
352          * @param dialog
353          */
354         private void prepareNotReadableFolderDialog(Dialog dialog) {
355
356               dialog.setTitle("[" + this.file.getName() + "]");
357
358         }
359
360         @Override
361         protected void onPrepareDialog(int id, Dialog dialog) {
362               switch (id) {
363               case FB_IMAGE_PREVIEW_DIALOG:
364                     prepareImagePreviewDialog(dialog);
365                     break;
366               case FB_NOT_AN_IMAGE_DIALOG:
```

```
367                   prepareNotAnImageDialog(dialog);
368                   break;
369             case FB_NOT_READABLE_FOLDER_DIALOG:
370                   prepareNotReadableFolderDialog(dialog);
371                   break;
372             default:
373                   break;
374             }
375       }
376
377       @Override
378       protected void onListItemClick(ListView l, View v, int position, long id) {
379
380             this.file = new File(path.get(position));
381
382             if (this.file.isDirectory()) {
383
384                   if (this.file.canRead()) {
385                         /*
386                          * Si es un directorio y se puede leer, entra en el directorio y
387                          * muestra el contenido de ese directorio
388                          */
389                         getDir(path.get(position));
390
391                   } else {
392                         /*
393                          * En caso de que el directorio no se pueda leer muestra un
394                          * mensaje de error
395                          */
396                         showDialog(FileBrowserActivity.FB_NOT_READABLE_FOLDER_DIALOG);
397                   }
398             } else {
399                   /*
400                    * Si no es un directorio, entonces será un archivo. Se verificará
401                    * si ese archivo es una imagen o no lo es.
402                    */
403                   String fileName = this.file.getName().toLowerCase();
404
405                   if ((fileName.endsWith(".jpg")) || (fileName.endsWith(".jpeg"))
406                               || (fileName.endsWith(".png"))
407                               || (fileName.endsWith(".bmp"))
408                               || (fileName.endsWith(".gif"))) {
409                         /*
410                          * En caso de que el archivo sea una imagen se mostrará un
411                          * cuadro de diálogo con una previsualización de la imagen para
412                          * que el usuario de la aplicación abra la imagen y verifique
413                          * que es la que quiere abrir
414                          */
415                         showDialog(FileBrowserActivity.FB_IMAGE_PREVIEW_DIALOG);
416                   } else {
417                         /*
418                          * En el caso el archivo no sea una imagen se mostrará
419                          * información sobre él en una ventana de diálogo
420                          */
```

```
                        showDialog(FileBrowserActivity.FB_NOT_AN_IMAGE_DIALOG);
                  }

            }
      }

      @Override
      public boolean onCreateOptionsMenu(Menu menu) {
            MenuInflater menuInflater = getMenuInflater();
            menuInflater.inflate(R.menu.file_browser_menu, menu);
            return true;
      }

      @Override
      public boolean onOptionsItemSelected(MenuItem item) {

            switch (item.getItemId()) {
            case R.id.file_browser_help_opt:
                  showDialog(FB_HELP_DIALOG);
                  return true;
            case R.id.file_browser_back_opt:
                  finish();
                  return true;
            default:
                  return super.onOptionsItemSelected(item);
            }

      }

      @Override
      public void onBackPressed() {
            finish();
      }
}
```

Listing D.8: **ColorMaskSelectionTouchView.java**

```
package com.imageRestoration.inpainting;

import android.app.ProgressDialog;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Matrix;
import android.os.Handler;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.widget.ImageView;

/**
 * Clase que heredando de ImageWiew extiende sus funcionalidades para poder
 * seleccionar colores que se encuentren en la imagen que alberga
 *
 * @author Samuel Martinez
 *
```

```
19  */
20 public class ColorMaskSelectionTouchView extends ImageView {
21
22       protected Handler handler;
23
24       private int maskColor = (140 << 24) | (00 << 16) | (255 << 8) | 00;
25
26       public ColorMaskSelectionTouchView(Context context) {
27             super(context);
28             handler = new Handler();
29       }
30
31       public ColorMaskSelectionTouchView(Context context, AttributeSet attrs) {
32             super(context, attrs);
33             handler = new Handler();
34       }
35
36       public ColorMaskSelectionTouchView(Context context, AttributeSet attrs,
37                   int defStyle) {
38             super(context, attrs, defStyle);
39             handler = new Handler();
40       }
41
42       @Override
43       protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
44             setMeasuredDimension(MeasureSpec.getSize(widthMeasureSpec),
45                         MeasureSpec.getSize(heightMeasureSpec));
46       }
47
48       /**
49        * Función que verifica si el punto introducido se encuentra o no dentro de
50        * la imagen
51        *
52        * @param point
53        * Vector bidimensional con el que se indica el punto introducido
54        * @param bitmapWidth
55        * Anchura del mapa de bits
56        * @param bitmapHeight
57        * Altura del mapa de bits
58        * @return Verdadero en caso de que el punto se encuentre dentro y falso en
59        * caso de que se encuentre fuera
60        */
61       protected boolean isInsideBitmap(float[] point, int bitmapWidth,
62                   int bitmapHeight) {
63
64             int x = Math.round(point[0]);
65             int y = Math.round(point[1]);
66
67             if ((x < 0) || (x > bitmapWidth) || (y < 0) || (y > bitmapHeight)) {
68                   return false;
69             } else {
70                   return true;
71             }
72
```

```
73        }
74
75        /**
76         * Función que verifica si el punto introducido se encuentra o no dentro de
77         * la imagen
78         *
79         * @param x
80         * Valor de la coordenada x del punto introducido
81         * @param y
82         * Valor de la coordenada y del punto introducido
83         * @param bitmapWidth
84         * Anchura del mapa de bits
85         * @param bitmapHeight
86         * Altura del mapa de bits
87         * @return Verdadero en caso de que el punto se encuentre dentro y falso en
88         * caso de que se encuentre fuera
89         */
90        protected boolean isInsideBitmap(int x, int y, int bitmapWidth,
91                    int bitmapHeight) {
92
93              if ((x < 0) || (x >= bitmapWidth) || (y < 0) || (y >= bitmapHeight)) {
94                    return false;
95              } else {
96                    return true;
97              }
98
99        }
100
101       @Override
102       public boolean onTouchEvent(final MotionEvent event) {
103
104             switch (event.getAction()) {
105             case MotionEvent.ACTION_DOWN:
106
107                   final ImageInpaintingColorSelectionActivity activity = (
                          ImageInpaintingColorSelectionActivity) getContext();
108
109                   float xTouched = event.getX();
110                   float yTouched = event.getY();
111                   float point[] = { xTouched, yTouched };
112
113                   Matrix transformMatrix = activity.getBitmapTransformMatrix();
114                   Matrix inverseMatrix = new Matrix();
115
116                   if (transformMatrix.invert(inverseMatrix)) {
117                         inverseMatrix.mapPoints(point);
118
119                         final Bitmap bitmap = activity.getBitmap();
120
121                         final int bitmapWidth = activity.getBitmapWidth();
122                         final int bitmapHeight = activity.getBitmapHeight();
123
124                         final int color = getColor(point, bitmap, bitmapWidth,
125                                     bitmapHeight);
```

```
126
127                     if (color != 0) {
128
129                           final ProgressDialog progressDialog = ProgressDialog
130                                       .show(getContext(), "",
131                                                   "Seleccionando información", true);
132
133                           Thread hilo = new Thread(new Runnable() {
134
135                                 public void run() {
136
137                                       for (int i = 0; i < bitmapWidth; i++) {
138                                             for (int j = 0; j < bitmapHeight; j++) {
139                                                   if (bitmap.getPixel(i, j) == color) {
140                                                         activity.setMaskPoint(i, j,
                                                               maskColor);
141                                                   }
142                                             }
143                                       }
144
145                                       handler.post(new Runnable() {
146
147                                             public void run() {
148                                                   activity.getMaskView().invalidate();
149                                             }
150                                       });
151                                       progressDialog.dismiss();
152
153                                 }
154                           });
155
156                           hilo.start();
157                     } else {
158                           Log.e(ColorMaskSelectionTouchView.class.toString()
159                                       + ".onTouchEvent()",
160                                       "El color del punto seleccionado a mostrar es 0 en
                                           el punto tocado"
161                                                   + xTouched + "," + yTouched
162                                                   + " en la imagen " + point[0] + ","
163                                                   + point[1]);
164                     }
165
166               } else {
167                     Log.e("", "La matriz de transformación no tiene inversa");
168               }
169
170               return true;
171         default:
172               return false;
173         }
174
175   }
176
177
```

```
178       /**
179        * Se obtiene el color en formato ARGB_8888 del punto (point) en caso de que se
              encuentre dentro de la imagen
180        * @param point Vector bidimensional donde se indican las coordenadas del punto
181        * @param bitmap Mapa de bits del que se quiere obtener el valor
182        * @param bitmapWidth Ancho del mapa de bits introducido
183        * @param bitmapHeight Alto del mapa de bits introducido
184        * @return Valor del color del píxel en formato ARGB_8888
185        */
186       protected int getColor(float[] point, Bitmap bitmap, int bitmapWidth,
187                   int bitmapHeight) {
188
189             int x = Math.round(point[0]);
190             int y = Math.round(point[1]);
191
192             if (isInsideBitmap(x, y, bitmapWidth, bitmapHeight)) {
193                   return bitmap.getPixel(x, y);
194             } else {
195                   return 0;
196             }
197
198       }
199
200 }
```

Listing D.9: **ColorSelectionMaskView.java**

```
1  package com.imageRestoration.inpainting;
2
3  import android.content.Context;
4  import android.graphics.Bitmap;
5  import android.graphics.Canvas;
6  import android.graphics.Matrix;
7  import android.util.AttributeSet;
8  import android.view.View;
9
10 /**
11  * Clase heredera de View que se utiliza para poder mostrar la máscara del color
12  * seleccionado
13  *
14  * @author Samuel Martinez
15  *
16  */
17 public class ColorSelectionMaskView extends View {
18
19       private ImageInpaintingColorSelectionActivity activity;
20
21       public ColorSelectionMaskView(Context context) {
22             super(context);
23
24             this.activity = (ImageInpaintingColorSelectionActivity) getContext();
25       }
26
27       public ColorSelectionMaskView(Context context, AttributeSet attrs) {
28             super(context, attrs);
```

```
29
30            this.activity = (ImageInpaintingColorSelectionActivity) getContext();
31      }
32
33      public ColorSelectionMaskView(Context context, AttributeSet attrs,
34                  int defStyle) {
35            super(context, attrs, defStyle);
36
37            this.activity = (ImageInpaintingColorSelectionActivity) getContext();
38      }
39
40      @Override
41      protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
42            setMeasuredDimension(MeasureSpec.getSize(widthMeasureSpec),
43                        MeasureSpec.getSize(heightMeasureSpec));
44      }
45
46      @Override
47      protected void onDraw(final Canvas canvas) {
48            super.onDraw(canvas);
49
50            Matrix imageMatrix = activity.getBitmapTransformMatrix();
51
52            Bitmap maskBitmap = activity.getMaskBitmap();
53
54            canvas.drawBitmap(maskBitmap, imageMatrix, null);
55
56      }
57
58  }
```

Listing D.10: **ImageInpaintingActivity.java**

```
1   package com.imageRestoration.inpainting;
2
3   import android.app.Activity;
4   import android.app.AlertDialog;
5   import android.app.Dialog;
6   import android.content.DialogInterface;
7   import android.content.Intent;
8   import android.graphics.Bitmap;
9   import android.os.Bundle;
10  import android.view.Gravity;
11  import android.view.LayoutInflater;
12  import android.view.Menu;
13  import android.view.MenuInflater;
14  import android.view.View;
15  import android.widget.Toast;
16
17  import com.imageRestoration.R;
18  import com.imageRestoration.util.ImageFileManager;
19
20  /**
21   * Clase que mapea la Actividad genérica para realizar el Inpainting
22   *
```

```
23  * @author Samu
24  *
25  */
26 public class ImageInpaintingActivity extends Activity {
27
28       /**
29        * Este bloque de código carga la librería nativa de C que contiene las
30        * funciones que se van a utilizar mediante JNI para mejorar el rendimiento
31        * del algoritmo
32        */
33       static {
34             // Carga de la librería nativa en C
35             String libName = "PhotoRestore";
36             System.loadLibrary(libName);
37       }
38
39       /**
40        * Firma de entrada de la función nativa que realiza el Inpainting
41        *
42        * @param inputPixels
43        * Píxeles de entrada
44        * @param inputMask
45        * Píxeles de entrada que indican la máscara
46        * @param width
47        * Ancho de la imagen
48        * @param height
49        * Alto de la imagen
50        * @param pixelsOutput
51        * Píxeles de salida (donde se devolverá el valor de la imagen
52        * filtrada) mediante un paso de parámetros por referencia
53        */
54       protected native void imageInpainting(int inputPixels[], int inputMask[],
55                   int width, int height, int pixelsOutput[]);
56
57       public final static int INPAINTING_HELP_DIALOG = 1;
58
59       private Bitmap bitmap;
60       protected int bitmapWidth, bitmapHeight;
61
62       protected int[] bitmapExtremes;
63
64       /* GETTERS Y SETTERS */
65
66       protected Bitmap getBitmap() {
67             return bitmap;
68       }
69
70       protected void setBitmap(Bitmap bitmap) {
71             this.bitmap = bitmap;
72             this.bitmapWidth = bitmap.getWidth();
73             this.bitmapHeight = bitmap.getHeight();
74       }
75
76       protected int getBitmapWidth() {
```

```
77              return bitmapWidth;
78        }
79
80        protected void setBitmapWidth(int bitmapWidth) {
81              this.bitmapWidth = bitmapWidth;
82        }
83
84        protected int getBitmapHeight() {
85              return bitmapHeight;
86        }
87
88        protected void setBitmapHeight(int bitmapHeight) {
89              this.bitmapHeight = bitmapHeight;
90        }
91
92        /**
93         * Muestra un mensaje flotante temporal que es utilizado para ayudar al
94         * usuario a utilizar la aplicación dando unas pequeñas indicaciones
95         */
96        protected void showToastSelectMask() {
97              // Se muestra un Toast con una cadena de texto que indica al usuario que
98              // debe seleccionar la máscara de inpainting
99              Toast mensajeToast = Toast.makeText(getApplicationContext(), "",
100                           Toast.LENGTH_LONG);
101             if (this instanceof ImageInpaintingMaskSelectionActivity) {
102                   mensajeToast.setText(R.string.inpaint_user_select_mask);
103             } else if (this instanceof ImageInpaintingColorSelectionActivity) {
104                   mensajeToast.setText(R.string.inpaint_user_select_color);
105             }
106             mensajeToast.setGravity(Gravity.CENTER, 0, 0);
107
108             mensajeToast.show();
109       }
110
111       /**
112        * Envío de resultados cuando la restauración se ha realizado correctamente
113        */
114       public void onRestorationSuccess() {
115             // Guardado del bitmap en memoria
116             String filePath = ImageFileManager.baseFolderPath.concat("temp.png");
117             if (ImageFileManager.saveBitmap(bitmap, "temp",
118                           Bitmap.CompressFormat.PNG)) {
119                   Intent intent = new Intent();
120                   Bundle extras = new Bundle();
121                   extras.putString("IMAGE_PATH_RESULT", filePath);
122                   intent.putExtras(extras);
123
124                   setResult(RESULT_OK, intent);
125                   finish();
126             } else {
127                   onCancelRestoration();
128             }
129       }
130
```

```
131         /**
132          * Envío de resultados cuando se cancela el proceso de restauración
133          */
134         protected void onCancelRestoration() {
135               setResult(RESULT_CANCELED);
136               finish();
137         }
138
139         /**
140          * Crea la ventana de diálogo y especifica las operaciones a realizar con
141          * ella
142          *
143          * @return Ventana de diálogo con instrucciones de ayuda para el filtrado de
144          * difusión anisótropa
145          */
146         protected Dialog createHelpDialog() {
147
148               // Creación de las variable para albergar la ventana de diálogo
149               AlertDialog.Builder builder = new AlertDialog.Builder(this);
150               Dialog dialog;
151
152               // Toma de la configuración visual del XML que mapea la ventana de
153               // diálogo y creación de la disposición
154               LayoutInflater factory = LayoutInflater.from(this);
155               View layout = null;
156               if (this instanceof ImageInpaintingMaskSelectionActivity) {
157                     layout = factory.inflate(R.layout.inpaint_mask_help_dialog, null);
158               } else if (this instanceof ImageInpaintingColorSelectionActivity) {
159                     layout = factory.inflate(R.layout.inpaint_color_help_dialog, null);
160               }
161               // Creación del listener para el botón "Aceptar"
162               DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                      OnClickListener() {
163
164                     public void onClick(DialogInterface dialog, int which) {
165
166                           dialog.dismiss();
167
168                     }
169               };
170
171               // Establecimiento de los valores de la ventana de diálogo y creación
172               builder = new AlertDialog.Builder(this);
173               builder.setIcon(android.R.drawable.ic_menu_help);
174               builder.setTitle(getString(R.string.help));
175               builder.setView(layout);
176               builder.setPositiveButton(R.string.accept, positiveButtonListener);
177               dialog = builder.create();
178
179               return dialog;
180         }
181
182         @Override
183         protected Dialog onCreateDialog(int id) {
```

```
184             Dialog dialog;
185
186             switch (id) {
187             case INPAINTING_HELP_DIALOG:
188                   dialog = createHelpDialog();
189                   break;
190             default:
191                   dialog = null;
192                   break;
193             }
194
195             return dialog;
196       }
197
198       @Override
199       public boolean onCreateOptionsMenu(Menu menu) {
200             MenuInflater menuInflater = getMenuInflater();
201             menuInflater.inflate(R.menu.inpainting_selection_menu, menu);
202             return true;
203       }
204
205       @Override
206       public void onBackPressed() {
207             onCancelRestoration();
208       }
209
210 }
```

Listing D.11: **ImageInpaintingColorSelectionActivity.java**

```
1  package com.imageRestoration.inpainting;
2
3  import android.app.ProgressDialog;
4  import android.content.res.Configuration;
5  import android.graphics.Bitmap;
6  import android.graphics.Canvas;
7  import android.graphics.Matrix;
8  import android.os.Bundle;
9  import android.util.Log;
10 import android.view.MenuItem;
11 import android.view.View;
12 import android.view.ViewTreeObserver;
13 import android.view.ViewTreeObserver.OnGlobalLayoutListener;
14
15 import com.imageRestoration.R;
16 import com.imageRestoration.showImage.ShowImageActivity;
17 import com.imageRestoration.util.ImageFileManager;
18
19 /**
20  * Clase heredera de ImageInpaintingActivity. Utilizada para crear la acción de
21  * la selección de máscara mediante el color de un píxel
22  *
23  * @author Samuel Martinez
24  *
25  */
```

```java
public class ImageInpaintingColorSelectionActivity extends
            ImageInpaintingActivity {

      // Variables que contienen la imagen y la máscara para procesar el
      // Inpainting
      private ColorMaskSelectionTouchView touchView;
      private ColorSelectionMaskView maskView;

      // Variable que contiene el mapa de bits de la máscara
      private Bitmap maskBitmap;

      /* GETTERS Y SETTERS */

      public Bitmap getMaskBitmap() {
            return this.maskBitmap;
      }

      public void setMaskPoint(int x, int y, int color) {
            this.maskBitmap.setPixel(x, y, color);
      }

      public ColorSelectionMaskView getMaskView() {
            return this.maskView;
      }

      @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.inpaint_color_selection);

            // Se toma la imagen pasada desde el Activity anterior y añade el bitmap
            // al TouchView
            Bundle bundle = getIntent().getExtras();
            String imagePath = bundle.getString("IMAGE_PATH");

            Bitmap bitmap = ImageFileManager.loadBitmapARGB_8888(imagePath, true);

            if (bitmap == null) {
                  Log.e(ImageInpaintingColorSelectionActivity.class.toString(),
                              "La imagen " + imagePath + " no ha sido cargada");
                  onCancelRestoration();
            } else {

                  this.setBitmap(bitmap);

                  this.maskBitmap = Bitmap.createBitmap(bitmapWidth, bitmapHeight,
                              Bitmap.Config.ARGB_8888);

                  // Se introduce la imagen, despues de transformarla en un mapa de
                  // bits
                  // dentro de su elemento en la vista
                  this.touchView = (ColorMaskSelectionTouchView) findViewById(R.id.
                      inpaint_color_selection_background);
                  this.touchView.setImageBitmap(bitmap);
```

```
79                this.maskView = (ColorSelectionMaskView) findViewById(R.id.
                      inpaint_color_mask_view);
80
81                ViewTreeObserver vto = touchView.getViewTreeObserver();
82                vto.addOnGlobalLayoutListener(new OnGlobalLayoutListener() {
83                      public void onGlobalLayout() {
84
85                            int bitmapWidth = getBitmapWidth();
86                            int bitmapHeight = getBitmapHeight();
87                            float[] bitmapExtremesFloat = new float[4];
88                            bitmapExtremesFloat[0] = 0;
89                            bitmapExtremesFloat[1] = 0;
90                            bitmapExtremesFloat[2] = bitmapWidth;
91                            bitmapExtremesFloat[3] = bitmapHeight;
92
93                            Matrix matrizTransformacion = getBitmapTransformMatrix();
94                            if (matrizTransformacion != null) {
95
96                                  matrizTransformacion.mapPoints(bitmapExtremesFloat);
97
98                                  bitmapExtremes = new int[bitmapExtremesFloat.length];
99                                  for (int i = 0; i < bitmapExtremesFloat.length; i++) {
100                                       bitmapExtremes[i] = Math
101                                                 .round(bitmapExtremesFloat[i]);
102                                 }
103
104                           }
105
106                           // Una vez utilizado se elimina el listener global
107                           touchView.getViewTreeObserver()
108                                       .removeGlobalOnLayoutListener(this);
109
110                     }
111               });
112
113               showToastSelectMask();
114
115         }
116
117   }
118
119   /**
120    * Puntos de los extremos de la imagen
121    *
122    * @return Puntos que mapean los extremos de la imagen
123    */
124   protected int[] getBitmapExtremes() {
125         return bitmapExtremes;
126   }
127
128   /**
129    * Crea una imagen de bits a partir de una vista view que se muestra por
130    * pantalla. El tamaño del mapa de bits devuelto será el mismo que el de la
131    * imagen que se muestra de fondo
```

```
132         *
133         * @param view
134         * Parámetro indicativo de la vista de la que se quiere obtener
135         * el mapa de bits
136         * @return Mapa de bits reproduciendo la vista view
137         */
138        protected Bitmap getMaskBitmap(View view) {
139
140              Matrix matrizTransformacion = getBitmapTransformMatrix();
141              float[] values = new float[9];
142              if (matrizTransformacion != null) {
143                    matrizTransformacion.getValues(values);
144                    Log.i(ImageInpaintingMaskSelectionActivity.class.toString(),
145                                "La matriz tiene valores: "
146                                            + matrizTransformacion.toString());
147              }
148
149              Bitmap bitmapScreenSize = Bitmap.createBitmap(
150                          Math.round(getBitmapWidth() * values[0]),
151                          Math.round(getBitmapHeight() * values[4]),
152                          Bitmap.Config.ARGB_8888);
153              Canvas canvas = new Canvas(bitmapScreenSize);
154
155              int[] bitmapExtremes = getBitmapExtremes();
156
157              view.layout(bitmapExtremes[0], bitmapExtremes[1], bitmapExtremes[2],
158                          bitmapExtremes[3]);
159              view.draw(canvas);
160
161              Bitmap bitmapRealSize = Bitmap.createScaledBitmap(bitmapScreenSize,
162                          getBitmapWidth(), getBitmapHeight(), false);
163
164              return bitmapRealSize;
165        }
166
167        @Override
168        public void onConfigurationChanged(Configuration newConfig) {
169              super.onConfigurationChanged(newConfig);
170              setContentView(R.layout.inpaint_color_selection);
171
172              Bitmap bitmap = this.getBitmap();
173
174              if (bitmap == null) {
175                    Log.e(ShowImageActivity.class.toString() + ".onCreate()",
176                                "La imagen no ha sido cargada");
177                    finish();
178              } else {
179
180                    this.touchView = null;
181                    this.touchView = (ColorMaskSelectionTouchView) findViewById(R.id.
                          inpaint_color_selection_background);
182                    this.maskView = (ColorSelectionMaskView) findViewById(R.id.
                          inpaint_color_mask_view);
183
```

```
184                 this.touchView.setImageBitmap(bitmap);
185
186                 this.maskBitmap = Bitmap.createBitmap(bitmapWidth, bitmapHeight,
187                             Bitmap.Config.ARGB_8888);
188                 this.maskView.invalidate();
189
190           }
191     }
192
193     @Override
194     public boolean onOptionsItemSelected(MenuItem item) {
195
196           switch (item.getItemId()) {
197           case R.id.inpainting_accept_opt:
198                 final ProgressDialog pd = ProgressDialog.show(this, "Inpainting",
199                             "Procesando imagen");
200
201                 new Thread(new Runnable() {
202
203                       public void run() {
204
205                             Bitmap bitmap = getBitmap();
206                             int width = bitmap.getWidth();
207                             int height = bitmap.getHeight();
208
209                             int pixelsInput[] = new int[width * height];
210                             bitmap.getPixels(pixelsInput, 0, width, 0, 0, width, height);
211
212                             int[] inputMask = new int[width * height];
213                             maskBitmap.getPixels(inputMask, 0, width, 0, 0, width,
214                                         height);
215
216                             int pixelsOutput[] = new int[width * height];
217
218                             // Launch image inpainting process
219
220                             final long t_init = System.currentTimeMillis();
221
222                             imageInpainting(pixelsInput, inputMask, width, height,
223                                         pixelsOutput);
224
225                             final long t_end = System.currentTimeMillis();
226
227                             bitmap = bitmap.copy(Bitmap.Config.ARGB_8888, true);
228                             bitmap.setPixels(pixelsOutput, 0, width, 0, 0, width,
229                                         height);
230
231                             setBitmap(bitmap);
232
233                             Log.i(ImageInpaintingColorSelectionActivity.class
234                                         .toString() + ".onOptionsItemSelected()",
235                                         "TIEMPO EMPLEADO EN EL INPAINTING:"
236                                                     + (t_end - t_init) / 1000 + "segundos");
237
```

```
                                pd.dismiss();

                                // Se finaliza la ejecución del algoritmo enviándolo
                                // de vuelta al activity que lo lanzó
                                onRestorationSuccess();
                          }
                    }).start();
                    return true;
              case R.id.inpainting_reset_opt:
                    this.maskBitmap = Bitmap.createBitmap(bitmapWidth, bitmapHeight,
                                Bitmap.Config.ARGB_8888);
                    this.maskView.invalidate();
                    return true;
              case R.id.inpainting_help_opt:
                    showDialog(INPAINTING_HELP_DIALOG);
                    return true;
              case R.id.inpainting_back_opt:
                    onCancelRestoration();
                    return true;
              default:
                    return super.onOptionsItemSelected(item);
              }
        }

        /**
         * Toda imagen insertada dentro un objeto de tipo ImageView se redimensiona
         * mediante una matriz de transformación, la cual se obtiene mediante este
         * método
         *
         * @return Matriz de transformación de la imagen
         */
        protected Matrix getBitmapTransformMatrix() {
              return this.touchView.getImageMatrix();
        }
}
```

Listing D.12: **ImageInpaintingMaskSelectionActivity.java**

```
package com.imageRestoration.inpainting;

import android.app.ProgressDialog;
import android.content.res.Configuration;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.os.Bundle;
import android.util.Log;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewTreeObserver;
import android.view.ViewTreeObserver.OnGlobalLayoutListener;
import android.widget.ImageView;

import com.imageRestoration.R;
import com.imageRestoration.showImage.ShowImageActivity;
```

```java
import com.imageRestoration.util.ImageFileManager;

/**
 * Clase heredera de ImageInpaintingActivity. Utilizada para crear la acción de
 * la selección de máscara mediante la selección manual realizada por el usuario
 *
 * @author Samuel Martinez
 *
 */
public class ImageInpaintingMaskSelectionActivity extends
            ImageInpaintingActivity {

      // Variables que contienen la imagen y la máscara para procesar el
      // Inpainting
      ImageView imageView;
      ManualMaskTouchView touchView;

      @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.inpaint_mask_selection);

            // Se toma la imagen pasada desde el Activity anterior y añade el bitmap
            // al TouchView
            Bundle bundle = getIntent().getExtras();
            String imagePath = bundle.getString("IMAGE_PATH");

            Bitmap bitmap = ImageFileManager.loadBitmapARGB_8888(imagePath, true);

            if (bitmap == null) {
                  Log.e(ImageInpaintingMaskSelectionActivity.class.toString(),
                              "La imagen " + imagePath + " no ha sido cargada");
                  onCancelRestoration();
            } else {

                  this.setBitmap(bitmap);

                  // Se introduce la imagen, despues de transformarla en un mapa de
                  // bits
                  // dentro de su elemento en la vista
                  this.imageView = (ImageView) findViewById(R.id.
                       inpaint_manual_selection_background);
                  this.imageView.setImageBitmap(getBitmap());
                  this.touchView = (ManualMaskTouchView) findViewById(R.id.
                       inpaint_manual_mask_view);

                  ViewTreeObserver vto = imageView.getViewTreeObserver();
                  vto.addOnGlobalLayoutListener(new OnGlobalLayoutListener() {
                        public void onGlobalLayout() {

                              int bitmapWidth = getBitmapWidth();
                              int bitmapHeight = getBitmapHeight();
                              float[] bitmapExtremesFloat = new float[4];
                              bitmapExtremesFloat[0] = 0;
```

```
70                              bitmapExtremesFloat[1] = 0;
71                              bitmapExtremesFloat[2] = bitmapWidth;
72                              bitmapExtremesFloat[3] = bitmapHeight;
73
74                              Matrix matrizTransformacion = getBitmapTransformMatrix();
75                              if (matrizTransformacion != null) {
76
77                                    matrizTransformacion.mapPoints(bitmapExtremesFloat);
78
79                                    bitmapExtremes = new int[bitmapExtremesFloat.length];
80                                    for (int i = 0; i < bitmapExtremesFloat.length; i++) {
81                                          bitmapExtremes[i] = Math
82                                                      .round(bitmapExtremesFloat[i]);
83                                    }
84
85                              }
86
87                              // Una vez utilizado se elimina el listener global
88                              imageView.getViewTreeObserver()
89                                          .removeGlobalOnLayoutListener(this);
90
91                        }
92                  });
93
94                  showToastSelectMask();
95
96            }
97
98      }
99
100     /**
101      * Puntos de los extremos de la imagen
102      *
103      * @return Puntos que mapean los extremos de la imagen
104      */
105     protected int[] getBitmapExtremes() {
106           return bitmapExtremes;
107     }
108
109     /**
110      * Toda imagen insertada dentro un objeto de tipo ImageView se redimensiona
111      * mediante una matriz de transformación, la cual se obtiene mediante este
112      * método
113      *
114      * @return Matriz de transformación de la imagen
115      */
116     protected Matrix getBitmapTransformMatrix() {
117           return this.imageView.getImageMatrix();
118     }
119
120     /**
121      * Crea una imagen de bits a partir de una vista view que se muestra por
122      * pantalla. El tamaño del mapa de bits devuelto será el mismo que el de la
123      * imagen que se muestra de fondo
```

```
124         *
125         * @param view
126         * Parámetro indicativo de la vista de la que se quiere obtener
127         * el mapa de bits
128         * @return Mapa de bits reproduciendo la vista view
129         */
130        protected Bitmap getMaskBitmap(View view) {
131
132              Matrix matrizTransformacion = getBitmapTransformMatrix();
133              float[] values = new float[9];
134              if (matrizTransformacion != null) {
135                    matrizTransformacion.getValues(values);
136                    Log.i(ImageInpaintingMaskSelectionActivity.class.toString(),
137                                "La matriz tiene valores: "
138                                      + matrizTransformacion.toString());
139              }
140
141              Bitmap bitmapScreenSize = Bitmap.createBitmap(
142                          Math.round(getBitmapWidth() * values[0]),
143                          Math.round(getBitmapHeight() * values[4]),
144                          Bitmap.Config.ARGB_8888);
145              Canvas canvas = new Canvas(bitmapScreenSize);
146
147              int[] bitmapExtremes = getBitmapExtremes();
148
149              view.layout(bitmapExtremes[0], bitmapExtremes[1], bitmapExtremes[2],
150                          bitmapExtremes[3]);
151              view.draw(canvas);
152
153              Bitmap bitmapRealSize = Bitmap.createScaledBitmap(bitmapScreenSize,
154                          getBitmapWidth(), getBitmapHeight(), false);
155
156              return bitmapRealSize;
157        }
158
159        @Override
160        public void onConfigurationChanged(Configuration newConfig) {
161              super.onConfigurationChanged(newConfig);
162              setContentView(R.layout.inpaint_mask_selection);
163
164              Bitmap bitmap = this.getBitmap();
165
166              if (bitmap == null) {
167                    Log.e(ShowImageActivity.class.toString() + ".onCreate()",
168                                "La imagen no ha sido cargada");
169                    finish();
170              } else {
171
172                    // Se introduce la imagen, despues de transformarla en un mapa de
173                    // bits
174                    // dentro de su elemento en la vista
175                    this.imageView = (ImageView) findViewById(R.id.
                         inpaint_manual_selection_background);
```

```
176                 this.touchView = (ManualMaskTouchView) findViewById(R.id.
                        inpaint_manual_mask_view);
177
178                 this.imageView.setImageBitmap(bitmap);
179
180                 this.touchView.resetPath();
181                 this.touchView.invalidate();
182
183           }
184     }
185
186     @Override
187     public boolean onOptionsItemSelected(MenuItem item) {
188
189           switch (item.getItemId()) {
190           case R.id.inpainting_accept_opt:
191                 final ProgressDialog pd = ProgressDialog.show(this, "Inpainting",
192                             "Procesando imagen");
193
194                 final Bitmap maskViewBitmap = getMaskBitmap(touchView);
195                 // TODO: Una vez establecidos los valores se debe llamar al método
196                 // nativo de inpainting
197
198                 new Thread(new Runnable() {
199
200                       public void run() {
201
202                             Bitmap bitmap = getBitmap();
203                             int width = bitmap.getWidth();
204                             int height = bitmap.getHeight();
205
206                             int pixelsInput[] = new int[width * height];
207                             bitmap.getPixels(pixelsInput, 0, width, 0, 0, width, height);
208
209                             int[] inputMask = new int[width * height];
210                             maskViewBitmap.getPixels(inputMask, 0, width, 0, 0, width,
211                                         height);
212                             int pixelsOutput[] = new int[width * height];
213
214                             // Launch image inpainting process
215
216                             final long t_init = System.currentTimeMillis();
217
218                             imageInpainting(pixelsInput, inputMask, width, height,
219                                         pixelsOutput);
220
221                             final long t_end = System.currentTimeMillis();
222
223                             bitmap = bitmap.copy(Bitmap.Config.ARGB_8888, true);
224                             bitmap.setPixels(pixelsOutput, 0, width, 0, 0, width,
225                                         height);
226
227                             setBitmap(bitmap);
228
```

```
                                Log.i(ImageInpaintingColorSelectionActivity.class
                                            .toString() + ".onOptionsItemSelected()",
                                            "TIEMPO EMPLEADO EN EL INPAINTING:"
                                                        + (t_end - t_init) / 1000 + "segundos");

                                pd.dismiss();

                                // Se finaliza la ejecución del algoritmo enviándolo
                                // de vuelta al activity que lo lanzó
                                onRestorationSuccess();
                          }
                    }).start();

                    return true;
              case R.id.inpainting_reset_opt:
                    this.touchView.resetPath();
                    this.touchView.invalidate();
                    return true;
              case R.id.inpainting_help_opt:
                    showDialog(INPAINTING_HELP_DIALOG);
                    return true;
              case R.id.inpainting_back_opt:
                    onCancelRestoration();
                    return true;
              default:
                    return super.onOptionsItemSelected(item);
              }
        }

}
```

Listing D.13: **ManualMaskTouchView.java**

```
package com.imageRestoration.inpainting;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;

/**
 * Clase heredera de View que se utiliza para pintar en su superficie la máscara
 * de Inpainting de forma manual
 *
 * @author Samuel Martinez
 *
 */
public class ManualMaskTouchView extends View {

      // Referencia al activity que la utiliza
      private ImageInpaintingMaskSelectionActivity activity;
```

```
23
24      // Variables que almacenan las coordenadas actual y anterior pulsadas
25      float previousX;
26      float previousY;
27
28      float currentX;
29      float currentY;
30
31      // Declaración del camino por donde pulsa el usuario y que hay que dibujar
32      Path path;
33      // Declaración del pincel
34      Paint paint;
35
36      public ManualMaskTouchView(Context context) {
37            super(context);
38
39            init();
40      }
41
42      public ManualMaskTouchView(Context context, AttributeSet attrs) {
43            super(context, attrs);
44
45            init();
46      }
47
48      public ManualMaskTouchView(Context context, AttributeSet attrs, int defStyle) {
49            super(context, attrs, defStyle);
50
51            init();
52      }
53
54      /**
55       * Función de inicialización de las variables de la clase
56       */
57      private void init() {
58            path = new Path();
59
60            activity = (ImageInpaintingMaskSelectionActivity) getContext();
61
62            paint = new Paint(Paint.ANTI_ALIAS_FLAG);
63            paint.setStyle(Paint.Style.STROKE);
64            paint.setStrokeWidth(10);
65            paint.setColor(Color.GREEN);
66            paint.setAlpha(150);
67      }
68
69      /**
70       * Función que reinicia el camino por el que el usuario ha pasado el dedo
71       */
72      public void resetPath() {
73            this.path.reset();
74      }
75
76      /**
```

```
77          * Método mediante el que se acotan la región donde puede "pintar" el usuario
78          *
79          * @param canvas
80          * Lienzo sobre el que se va a pintar
81          */
82         public void onDraw(Canvas canvas) {
83
84               int[] bitmapExtremes = activity.getBitmapExtremes();
85
86               this.layout(bitmapExtremes[0], bitmapExtremes[1], bitmapExtremes[2],
87                           bitmapExtremes[3]);
88
89               canvas.drawPath(path, paint);
90         }
91
92         @Override
93         public boolean onTouchEvent(final MotionEvent event) {
94
95               currentX = event.getX();
96               currentY = event.getY();
97
98               switch (event.getAction()) {
99               case MotionEvent.ACTION_DOWN:
100                    path.moveTo(currentX, currentY);
101                    break;
102              case MotionEvent.ACTION_MOVE:
103                    path.quadTo(previousX, previousY, currentX, currentY);
104                    break;
105              case MotionEvent.ACTION_UP:
106                    path.quadTo(previousX, previousY, currentX, currentY);
107                    break;
108              }
109
110              previousX = currentX;
111              previousY = currentY;
112
113              postInvalidate();
114
115              return true;
116
117        }
118
119 }
```

Listing D.14: **MainScreenActivity.java**

```
1  /**
2   *
3   */
4  package com.imageRestoration.mainScreen;
5
6  import android.app.Activity;
7  import android.app.AlertDialog;
8  import android.app.Dialog;
9  import android.content.DialogInterface;
```

```
10 import android.content.Intent;
11 import android.os.Bundle;
12 import android.view.LayoutInflater;
13 import android.view.Menu;
14 import android.view.MenuInflater;
15 import android.view.MenuItem;
16 import android.view.View;
17 import android.view.View.OnClickListener;
18 import android.widget.Button;
19
20 import com.imageRestoration.R;
21 import com.imageRestoration.fileBrowser.FileBrowserActivity;
22
23 /**
24  * Clase que genera la pantalla inicial de la aplicación
25  *
26  * @author Samuel Martinez
27  *
28  */
29 public class MainScreenActivity extends Activity {
30
31       private static final int MS_ABOUT_APP_DIALOG = 1;
32       private static final int MS_HELP_DIALOG = 4;
33       private static final int MS_EXIT_DIALOG = 5;
34
35       private Button openFileButton;
36       private Button aboutAppButton;
37
38       @Override
39       public void onCreate(Bundle savedInstanceState) {
40             super.onCreate(savedInstanceState);
41             setContentView(R.layout.main_screen);
42
43             /*
44              * Config for "Open file" button
45              */
46             openFileButton = (Button) findViewById(R.id.open_file_button);
47
48             OnClickListener listenerOpenFileButton = new OnClickListener() {
49
50                   public void onClick(View v) {
51                         Intent intent = new Intent(MainScreenActivity.this,
52                                     FileBrowserActivity.class);
53                         startActivity(intent);
54                   }
55             };
56
57             openFileButton.setOnClickListener(listenerOpenFileButton);
58
59             /*
60              * Config for "About..." button
61              */
62             aboutAppButton = (Button) findViewById(R.id.about_app_button);
63
```

```
64              OnClickListener listenerAboutAppButton = new OnClickListener() {
65                    public void onClick(View v) {
66                          showDialog(MainScreenActivity.MS_ABOUT_APP_DIALOG);
67                    }
68              };
69
70              aboutAppButton.setOnClickListener(listenerAboutAppButton);
71
72        }
73
74        /**
75         * Método que crea la ventana de diálogo con información sobre la aplicación
76         * y sus acciones correspondientes
77         *
78         * @return Ventana de diálogo
79         */
80        private Dialog createAboutDialog() {
81
82              AlertDialog.Builder builder;
83              AlertDialog alertDialog = null;
84
85              LayoutInflater factory = LayoutInflater.from(this);
86              final View layout = factory.inflate(R.layout.about_app_dialog, null);
87
88              builder = new AlertDialog.Builder(this);
89              builder.setTitle(R.string.about_app);
90              builder.setView(layout);
91              builder.setPositiveButton(R.string.accept, null);
92              alertDialog = builder.create();
93
94              return alertDialog;
95        }
96
97        /**
98         * Método que crea la ventana de diálogo de ayuda general de la aplicación y
99         * sus acciones correspondientes
100        *
101        * @return Ventana de diálogo
102        */
103       private Dialog createHelpDialog() {
104             // Creación de las variable para albergar la ventana de diálogo
105             AlertDialog.Builder builder = new AlertDialog.Builder(this);
106             Dialog dialog;
107
108             // Toma de la configuración visual del XML que mapea la ventana de
109             // diálogo y creación de la disposición
110             LayoutInflater factory = LayoutInflater.from(this);
111             final View layout = factory.inflate(R.layout.main_screen_help_dialog,
112                         null);
113
114             // Creación del listener para el botón "Cancelar"
115             DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                    OnClickListener() {
116
```

```
117                 public void onClick(DialogInterface dialog, int which) {
118
119                       dialog.dismiss();
120
121                 }
122           };
123
124           // Establecimiento de los valores de la ventana de diálogo y creación
125           builder = new AlertDialog.Builder(this);
126           builder.setIcon(android.R.drawable.ic_menu_help);
127           builder.setTitle(getString(R.string.help));
128           builder.setView(layout);
129           builder.setPositiveButton(R.string.accept, positiveButtonListener);
130           dialog = builder.create();
131
132           return dialog;
133     }
134
135     /**
136      * Método que crea la ventana de diálogo para cerrar la aplicación y sus
137      * acciones correspondientes
138      *
139      * @return Ventana de diálogo
140      */
141     private Dialog createExitDialog() {
142           // Creación de las variable para albergar la ventana de diálogo
143           AlertDialog.Builder builder = new AlertDialog.Builder(this);
144           Dialog dialog;
145
146           // Toma de la configuración visual del XML que mapea la ventana de
147           // diálogo y creación de la disposición
148           LayoutInflater factory = LayoutInflater.from(this);
149           final View layout = factory.inflate(R.layout.exit_dialog, null);
150
151           // Creación del listener para el botón "Aceptar"
152           DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                OnClickListener() {
153
154                 public void onClick(DialogInterface dialog, int which) {
155
156                       // android.os.Process.killProcess(android.os.Process.myPid());
157                       finish();
158
159                 }
160           };
161
162           // Creación del listener para el botón "Cancelar"
163           DialogInterface.OnClickListener negativeButtonListener = new DialogInterface.
                OnClickListener() {
164
165                 public void onClick(DialogInterface dialog, int which) {
166
167                       dialog.dismiss();
168
```

```
169                 }
170           };
171
172           // Establecimiento de los valores de la ventana de diálogo y creación
173           builder = new AlertDialog.Builder(this);
174           builder.setIcon(android.R.drawable.ic_menu_close_clear_cancel);
175           builder.setTitle(getString(R.string.app_exit));
176           builder.setView(layout);
177           builder.setPositiveButton(R.string.accept, positiveButtonListener);
178           builder.setNegativeButton(R.string.cancel, negativeButtonListener);
179           dialog = builder.create();
180
181           return dialog;
182     }
183
184     @Override
185     protected Dialog onCreateDialog(int id) {
186           Dialog dialog;
187
188           switch (id) {
189           case MS_ABOUT_APP_DIALOG:
190                 dialog = createAboutDialog();
191                 break;
192           case MS_HELP_DIALOG:
193                 dialog = createHelpDialog();
194                 break;
195           case MS_EXIT_DIALOG:
196                 dialog = createExitDialog();
197                 break;
198           default:
199                 dialog = null;
200                 break;
201           }
202
203           return dialog;
204     }
205
206     @Override
207     public boolean onCreateOptionsMenu(Menu menu) {
208           MenuInflater menuInflater = getMenuInflater();
209           menuInflater.inflate(R.menu.main_screen_menu, menu);
210           return true;
211     }
212
213     @Override
214     public boolean onOptionsItemSelected(MenuItem item) {
215
216           switch (item.getItemId()) {
217           case R.id.general_help_opt:
218                 showDialog(MS_HELP_DIALOG);
219                 return true;
220           case R.id.exit_opt:
221                 showDialog(MS_EXIT_DIALOG);
222                 return true;
```

```
223             default:
224                   return super.onOptionsItemSelected(item);
225             }
226
227       }
228
229 }
```

Listing D.15: **ShowImageActivity.java**

```
1  /**
2   *
3   */
4  package com.imageRestoration.showImage;
5
6  import android.app.Activity;
7  import android.app.AlertDialog;
8  import android.app.Dialog;
9  import android.content.DialogInterface;
10 import android.content.Intent;
11 import android.content.res.Configuration;
12 import android.graphics.Bitmap;
13 import android.graphics.Bitmap.CompressFormat;
14 import android.os.Bundle;
15 import android.util.Log;
16 import android.view.LayoutInflater;
17 import android.view.Menu;
18 import android.view.MenuInflater;
19 import android.view.MenuItem;
20 import android.view.View;
21 import android.widget.ArrayAdapter;
22 import android.widget.EditText;
23 import android.widget.ImageView;
24 import android.widget.Spinner;
25 import android.widget.Toast;
26
27 import com.imageRestoration.R;
28 import com.imageRestoration.anisotropicDiffusion.AnisotropicDiffusionActivity;
29 import com.imageRestoration.inpainting.ImageInpaintingColorSelectionActivity;
30 import com.imageRestoration.inpainting.ImageInpaintingMaskSelectionActivity;
31 import com.imageRestoration.util.ImageFileManager;
32
33 /**
34  * Clase que mapea la actividad de gestión principal de la aplicación, dentro de
35  * la que se muestra la imagen que se quiere retocar o la imagen ya restaurada
36  *
37  * @author Samuel Martinez
38  *
39  */
40 public class ShowImageActivity extends Activity {
41
42       private final static int IM_INPAINT_DIALOG = 0;
43       private final static int IM_SAVE_DIALOG = 1;
44       private final static int IM_HELP_DIALOG = 2;
45       private final static int IM_BACK_FILE_BROWSER_DIALOG = 3;
```

```
46
47      // Cadena de texto que contiene la ruta al fichero que hay que cargar
48      private String filePath;
49      // Variable que contiene la imagen que se desea tratar
50      private Bitmap bitmap;
51
52      @Override
53      public void onCreate(Bundle savedInstanceState) {
54            super.onCreate(savedInstanceState);
55            setContentView(R.layout.show_image);
56
57            // Se toma la ruta pasada desde el Activity anterior y se crea un nuevo
58            // fichero que contiene la imagen
59            Bundle bundle = getIntent().getExtras();
60            filePath = bundle.getString("IMAGE_PATH");
61
62            this.bitmap = ImageFileManager.loadBitmapARGB_8888(filePath, false);
63
64            if (this.bitmap == null) {
65                  Log.e(ShowImageActivity.class.toString() + ".onCreate()",
66                              "La imagen " + filePath + " no ha sido cargada");
67                  finish();
68            } else {
69
70                  // Se introduce la imagen, despues de transformarla en un mapa de
71                  // bits
72                  // dentro de su elemento en la vista
73                  ImageView imageView = (ImageView) findViewById(R.id.prerestored_image);
74                  imageView.setImageBitmap(bitmap);
75
76            }
77      }
78
79      @Override
80      public void onActivityResult(int requestCode, int resultCode, Intent data) {
81            super.onActivityResult(requestCode, resultCode, data);
82
83            if (resultCode == Activity.RESULT_OK) {
84                  // Se toma la cadena de texto del elemento anterior al actual
85                  Bundle bundle = data.getExtras();
86                  this.filePath = bundle.getString("IMAGE_PATH_RESULT");
87
88                  this.bitmap = ImageFileManager.loadBitmapARGB_8888(filePath, true);
89
90                  if (this.bitmap == null) {
91                        Log.e(ShowImageActivity.class.toString()
92                                    + ".onActivityResult()", "La imagen " + filePath
93                                    + " no ha sido cargada");
94                        finish();
95                  } else {
96
97                        // Se pone la imagen resultado como la nueva imagen
98                        ImageView imageView = (ImageView) findViewById(R.id.prerestored_image)
                              ;
```

```
99                        imageView.setImageBitmap(bitmap);
100                       imageView.refreshDrawableState();
101
102                 }
103           }
104
105     }
106
107     @Override
108     public void onConfigurationChanged(Configuration newConfig) {
109           super.onConfigurationChanged(newConfig);
110           setContentView(R.layout.show_image);
111
112           if (this.bitmap == null) {
113                 Log.e(ShowImageActivity.class.toString() + ".onCreate()",
114                             "La imagen " + filePath + " no ha sido cargada");
115                 finish();
116           } else {
117
118                 ImageView imageView = (ImageView) findViewById(R.id.prerestored_image);
119                 imageView.setImageBitmap(bitmap);
120
121           }
122     }
123
124     /**
125      * Función que lanza la Actividad para seleccionar la máscara para
126      * Inpainting de forma manual
127      */
128     private void launchInpaintingManualSelectionActivity() {
129           String extension = ".png";
130           String fileName = "temp";
131           this.filePath = ImageFileManager.baseFolderPath;
132           if (ImageFileManager.saveBitmap(bitmap, fileName, CompressFormat.PNG)) {
133                 this.filePath = this.filePath.concat(fileName);
134                 this.filePath = this.filePath.concat(extension);
135                 Intent intent = new Intent(ShowImageActivity.this,
136                             ImageInpaintingMaskSelectionActivity.class);
137
138                 Bundle extras = new Bundle();
139                 extras.putString("IMAGE_PATH", this.filePath);
140                 intent.putExtras(extras);
141
142                 startActivityForResult(intent, 1);
143           } else {
144                 Log.e(ShowImageActivity.class.toString()
145                             + ".launchInpaintingManualSeletionActivity()",
146                             "El fichero " + this.filePath
147                                         + " no ha sido almacenado correctamente");
148           }
149
150     }
151
152     /**
```

```
153         * Función que lanza la Actividad para seleccionar la máscara para
154         * Inpainting mediante la selección del color de los píxeles
155         */
156        private void launchInpaintingColorSelectionActivity() {
157
158              String extension = ".png";
159              String fileName = "temp";
160              this.filePath = ImageFileManager.baseFolderPath;
161              if (ImageFileManager.saveBitmap(bitmap, fileName, CompressFormat.PNG)) {
162                    this.filePath = this.filePath.concat(fileName);
163                    this.filePath = this.filePath.concat(extension);
164                    Intent intent = new Intent(ShowImageActivity.this,
165                                ImageInpaintingColorSelectionActivity.class);
166
167                    Bundle extras = new Bundle();
168                    extras.putString("IMAGE_PATH", this.filePath);
169                    intent.putExtras(extras);
170
171                    startActivityForResult(intent, 1);
172              } else {
173                    Log.e(ShowImageActivity.class.toString()
174                                + ".launchInpaintingColorSelectionActivity()",
175                                "El fichero " + this.filePath
176                                            + " no ha sido almacenado correctamente");
177              }
178
179        }
180
181        /**
182         * Función que lanza la Actividad para seleccionar los parámetros para el
183         * filtrado de difusión anisótropa y realizar el filtrado
184         */
185        private void launchAnisotropicDiffusionActivity() {
186
187              String extension = ".png";
188              String fileName = "temp";
189              this.filePath = ImageFileManager.baseFolderPath;
190              if (ImageFileManager.saveBitmap(bitmap, fileName, CompressFormat.PNG)) {
191                    this.filePath = this.filePath.concat(fileName);
192                    this.filePath = this.filePath.concat(extension);
193                    Intent intent = new Intent(ShowImageActivity.this,
194                                AnisotropicDiffusionActivity.class);
195
196                    Bundle extras = new Bundle();
197                    extras.putString("IMAGE_PATH", this.filePath);
198                    intent.putExtras(extras);
199
200                    startActivityForResult(intent, 1);
201              } else {
202                    Log.e(ShowImageActivity.class.toString()
203                                + ".launchAnisotropicDiffusionActivity()", "El fichero "
204                                + this.filePath + " no ha sido almacenado correctamente");
205              }
206
```

```
207     }
208
209     /**
210      * Crea la ventana de diálogo mediante la cual decide el usuario si
211      * seleccionar la máscara de Inpainting de forma manual o mediante color
212      *
213      * @return Ventana de diálogo
214      */
215     private Dialog createInpaintDialog() {
216
217           AlertDialog.Builder builder;
218           AlertDialog alertDialog = null;
219
220           // Creación del listener para el botón
221           // "Inpainting por selección de máscara"
222           DialogInterface.OnClickListener maskSelectionButtonListener = new DialogInterface.
                OnClickListener() {
223
224                 public void onClick(DialogInterface dialog, int which) {
225
226                       launchInpaintingManualSelectionActivity();
227
228                 }
229           };
230
231           // Creación del listener para el botón
232           // "Inpainting por selección de color"
233           DialogInterface.OnClickListener colorSelectionButtonListener = new DialogInterface
                .OnClickListener() {
234
235                 public void onClick(DialogInterface dialog, int which) {
236                       launchInpaintingColorSelectionActivity();
237                 }
238           };
239
240           // Creación del listener para el botón "Cancelar"
241           DialogInterface.OnClickListener negativeButtonListener = new DialogInterface.
                OnClickListener() {
242
243                 public void onClick(DialogInterface dialog, int which) {
244
245                       dialog.dismiss();
246
247                 }
248           };
249
250           builder = new AlertDialog.Builder(this);
251           builder.setTitle(R.string.about_app);
252           builder.setMessage(R.string.inpaint_type_mask_selection_question);
253           builder.setPositiveButton(R.string.inpaint_mask_selection,
254                       maskSelectionButtonListener);
255           builder.setNeutralButton(R.string.inpaint_color_selection,
256                       colorSelectionButtonListener);
257           builder.setNegativeButton(R.string.cancel, negativeButtonListener);
```

```
258             alertDialog = builder.create();
259
260             return alertDialog;
261         }
262
263         /**
264          * Crea la ventana de diálogo y genera las acciones mediante las cuales
265          * decide cómo almancenar la imagen
266          *
267          * @return Ventana de diálogo
268          */
269         private Dialog createSaveDialog() {
270             // Creación de las variable para albergar la ventana de diálogo
271             AlertDialog.Builder builder = new AlertDialog.Builder(this);
272             Dialog dialog;
273
274             // Toma de la configuración visual del XML que mapea la ventana de
275             // diálogo y creación de la disposición
276             LayoutInflater factory = LayoutInflater.from(this);
277             final View layout = factory.inflate(R.layout.save_image_dialog, null);
278
279             // Insertar los posibles tipos de archivo que puede tomar el usuario
280             // para almacenar la imagen restaurada
281             final Spinner tiposImagen = (Spinner) layout
282                     .findViewById(R.id.image_type_spinner);
283             ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
284                     this, R.array.save_image_types,
285                     android.R.layout.simple_spinner_item);
286             adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
287             tiposImagen.setAdapter(adapter);
288
289             // Creación del listener para el botón "Aceptar"
290             DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                    OnClickListener() {
291
292                 public void onClick(DialogInterface dialog, int which) {
293
294                     // Extracción del nombre del fichero
295                     final EditText fileNameEditText = (EditText) layout
296                             .findViewById(R.id.new_file_name);
297                     String fileName = fileNameEditText.getText().toString();
298
299                     // Extracción del tipo del fichero y creación de la variable que
300                     // lo representa
301                     final Spinner spinner = (Spinner) layout
302                             .findViewById(R.id.image_type_spinner);
303                     String fileType = spinner.getSelectedItem().toString();
304                     Bitmap.CompressFormat compressFormat = null;
305                     if (fileType.equals("JPEG")) {
306                         compressFormat = Bitmap.CompressFormat.JPEG;
307                     } else if (fileType.equals("PNG")) {
308                         compressFormat = Bitmap.CompressFormat.PNG;
309                     }
310
```

```
311                     // Log.i(ShowImageActivity.class.toString(),
312                     // "Nombre del archivo " + fileName);
313                     // Log.i(ShowImageActivity.class.toString(), "Tipo del archivo "
314                     // + fileType);
315
316                     if (!ImageFileManager.checkFileName(fileName)) {
317                           // Si el nombre del archivo no es correcto, se muestra un
318                           // mensaje de error para que sea corregido
319                           Toast mensajeError = Toast.makeText(
320                                       getApplicationContext(),
321                                       getString(R.string.save_image_file_name_mistake),
322                                       Toast.LENGTH_LONG);
323                           mensajeError.show();
324
325                     } else if (compressFormat == null) {
326                           // Si no se ha seleccionado formato, se muestra un mensaje
327                           // para que se seleccione
328                           Toast mensajeError = Toast.makeText(
329                                       getApplicationContext(),
330                                       getString(R.string.save_image_file_type_mistake),
331                                       Toast.LENGTH_LONG);
332                           mensajeError.show();
333
334                     } else {
335                           // Se crea un mensaje con el que mostrar el resultado de la
336                           // acción
337                           Toast resultMessage = Toast.makeText(
338                                       getApplicationContext(),
339                                       getString(R.string.save_image_success),
340                                       Toast.LENGTH_LONG);
341
342                           // Se trata de guardar el archivo en memoria
343                           if (!ImageFileManager.saveBitmap(bitmap, fileName,
344                                       compressFormat)) {
345                                 resultMessage.setText(R.string.save_image_fail);
346                           }
347
348                           // Se muestra el mensaje al usuario
349                           resultMessage.show();
350
351                     }
352
353                     dialog.dismiss();
354
355               }
356           };
357
358           // Creación del listener para el botón "Cancelar"
359           DialogInterface.OnClickListener negativeButtonListener = new DialogInterface.
                  OnClickListener() {
360
361                public void onClick(DialogInterface dialog, int which) {
362
363                     dialog.dismiss();
```

```
364
365                 }
366           };
367
368           // Establecimiento de los valores de la ventana de diálogo y creación
369           builder = new AlertDialog.Builder(this);
370           builder.setIcon(android.R.drawable.ic_menu_save);
371           builder.setTitle(getString(R.string.save));
372           builder.setView(layout);
373           builder.setPositiveButton(R.string.accept, positiveButtonListener);
374           builder.setNegativeButton(R.string.cancel, negativeButtonListener);
375           dialog = builder.create();
376
377           return dialog;
378     }
379
380     /**
381      * Crea la ventana de diálogo donde se muestra una ayuda general para el
382      * tratamiento que se le puede dar a la imagen
383      *
384      * @return Ventana de diálogo
385      */
386     private Dialog createHelpDialog() {
387           // Creación de las variable para albergar la ventana de diálogo
388           AlertDialog.Builder builder = new AlertDialog.Builder(this);
389           Dialog dialog;
390
391           // Toma de la configuración visual del XML que mapea la ventana de
392           // diálogo y creación de la disposición
393           LayoutInflater factory = LayoutInflater.from(this);
394           final View layout = factory.inflate(R.layout.image_help_dialog, null);
395
396           // Creación del listener para el botón "Aceptar"
397           DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                OnClickListener() {
398
399                 public void onClick(DialogInterface dialog, int which) {
400
401                       dialog.dismiss();
402
403                 }
404           };
405
406           // Establecimiento de los valores de la ventana de diálogo y creación
407           builder = new AlertDialog.Builder(this);
408           builder.setIcon(android.R.drawable.ic_menu_help);
409           builder.setTitle(getString(R.string.help));
410           builder.setView(layout);
411           builder.setPositiveButton(R.string.accept, positiveButtonListener);
412           dialog = builder.create();
413
414           return dialog;
415     }
416
```

```
417       /**
418        * Crea la ventana de diálogo mediante la cual decide el usuario si quiere
419        * volver para cargar o no otra imagen. En caso de que el usuario cargue
420        * otra imagen, todos los progresos realizados con la que existe en la
421        * actualidad se perderán
422        *
423        * @return Ventana de diálogo
424        */
425       private Dialog createBackToFileBrowserDialog() {
426             // Creación de las variable para albergar la ventana de diálogo
427             AlertDialog.Builder builder = new AlertDialog.Builder(this);
428             Dialog dialog;
429
430             // Creación del listener para el botón "Aceptar"
431             DialogInterface.OnClickListener positiveButtonListener = new DialogInterface.
                      OnClickListener() {
432
433                   public void onClick(DialogInterface dialog, int which) {
434
435                         finish();
436
437                   }
438             };
439
440             // Creación del listener para el botón "Cancelar"
441             DialogInterface.OnClickListener negativeButtonListener = new DialogInterface.
                      OnClickListener() {
442
443                   public void onClick(DialogInterface dialog, int which) {
444
445                         dialog.dismiss();
446
447                   }
448             };
449
450             // Establecimiento de los valores de la ventana de diálogo y creación
451             builder = new AlertDialog.Builder(this);
452             builder.setIcon(android.R.drawable.ic_menu_close_clear_cancel);
453             builder.setTitle(getString(R.string.load_other_image));
454             builder.setMessage(R.string.load_other_image_dialog_msg);
455             builder.setPositiveButton(R.string.accept, positiveButtonListener);
456             builder.setNegativeButton(R.string.cancel, negativeButtonListener);
457             dialog = builder.create();
458
459             return dialog;
460       }
461
462       @Override
463       protected Dialog onCreateDialog(int id) {
464             Dialog dialog;
465
466             switch (id) {
467             case IM_INPAINT_DIALOG:
468                   dialog = createInpaintDialog();
```

```
469                 break;
470           case IM_SAVE_DIALOG:
471                 dialog = createSaveDialog();
472                 break;
473           case IM_HELP_DIALOG:
474                 dialog = createHelpDialog();
475                 break;
476           case IM_BACK_FILE_BROWSER_DIALOG:
477                 dialog = createBackToFileBrowserDialog();
478                 break;
479           default:
480                 dialog = null;
481                 break;
482           }
483
484           return dialog;
485     }
486
487     @Override
488     public boolean onCreateOptionsMenu(Menu menu) {
489           MenuInflater menuInflater = getMenuInflater();
490           menuInflater.inflate(R.menu.image_preview_menu, menu);
491           return true;
492     }
493
494     @Override
495     public boolean onOptionsItemSelected(MenuItem item) {
496
497           switch (item.getItemId()) {
498           case R.id.inpaint_opt:
499                 showDialog(IM_INPAINT_DIALOG);
500                 return true;
501           case R.id.denoise_opt:
502                 launchAnisotropicDiffusionActivity();
503                 return true;
504                 /*
505                  * case R.id.deblur_opt: launchDeblurActivity(); return true;
506                  */
507           case R.id.save_opt:
508                 showDialog(IM_SAVE_DIALOG);
509                 return true;
510           case R.id.image_help_opt:
511                 showDialog(IM_HELP_DIALOG);
512                 return true;
513           case R.id.load_other_image_opt:
514                 showDialog(IM_BACK_FILE_BROWSER_DIALOG);
515                 return true;
516           default:
517                 return super.onOptionsItemSelected(item);
518           }
519
520     }
521
522     @Override
```

```
523       public void onBackPressed() {
524             finish();
525       }
526
527 }
```

Listing D.16: **ImageFileManager.java**

```
1  package com.imageRestoration.util;
2
3  import java.io.File;
4  import java.io.FileOutputStream;
5
6  import android.graphics.Bitmap;
7  import android.graphics.BitmapFactory;
8  import android.util.Log;
9
10 /**
11  * Clase mediante la cual se realiza la gestión del almacenamiento de imágenes
12  * dentro de la tarjeta SD
13  *
14  * @author Samuel Martinez
15  *
16  */
17 public class ImageFileManager {
18
19       private final static String sdFolderPath = "/sdcard/";
20       private final static String appFolderPath = "PhotoRestore/";
21       public final static String baseFolderPath = sdFolderPath
22                   .concat(appFolderPath);
23
24       /**
25        * Verifica que la carpeta donde se almacenan las imágenes existe y, caso de
26        * que no exista, la crea
27        *
28        * @return Si la carpeta existe o si ha sido creada correctamente en caso de
29        * su inexistencia
30        */
31       private static boolean checkSaveFolder() {
32
33             File baseFolder = new File(baseFolderPath);
34
35             if (baseFolder.exists()) {
36                   Log.i(ImageFileManager.class.toString(), "La carpeta ["
37                               + baseFolder.toString() + "] ya existe");
38                   return true;
39             } else {
40                   Boolean successCreatingFolder = baseFolder.mkdirs();
41
42                   if (!successCreatingFolder) {
43                         Log.e(ImageFileManager.class.toString(), "La carpeta ["
44                                     + baseFolder + "] no ha podido ser creada");
45                   }
46                   return successCreatingFolder;
47             }
```

```
48
49          }
50
51          /**
52           * Verifica el contenido de la cadena de nombre de archivo. La verificación
53           * se realiza en relación a si posee caracteres incorrectos
54           *
55           * @param fileName
56           * Nombre del archivo con el cual se desea guardar
57           * @return Verificación de si contiene o no caracteres no permitidos
58           */
59          public static boolean checkFileName(String fileName) {
60
61                if (fileName.contains(",") || fileName.contains(".")
62                            || fileName.contains("\"") || fileName.contains("\'")
63                            || fileName.contains("\n") || fileName.contains("\\")
64                            || fileName.contains("/") || fileName.contains("|")
65                            || fileName.contains("#") || fileName.contains("@")
66                            || fileName.contains("$")) {
67                      return false;
68                }
69
70                return true;
71          }
72
73          /**
74           * Esta función se encarga de almacenar la imagen del bitmap en
75           *
76           * @param bitmap
77           * Mapa de bits a guardar en memoria
78           * @param fileName
79           * Nombre del archivo a crear
80           * @param bitmapFormat
81           * Formato en el que se va a almacenar el mapa de bits
82           * @return Un flag indicativo del resultado del proceso
83           */
84          public static boolean saveBitmap(Bitmap bitmap, String fileName,
85                      Bitmap.CompressFormat bitmapFormat) {
86
87                FileOutputStream outFile;
88                String fullFileName = null;
89                if (fileName.contains(baseFolderPath)) {
90                      fullFileName = fileName;
91                } else {
92                      fullFileName = baseFolderPath.concat(fileName);
93                }
94
95                // Comprobación de que el formato de archivo es correcto y adición del
96                // formato al final del nombre del archivo
97                if (bitmapFormat == Bitmap.CompressFormat.JPEG) {
98                      fullFileName = fullFileName.concat(".jpg");
99                } else if (bitmapFormat == Bitmap.CompressFormat.PNG) {
100                     fullFileName = fullFileName.concat(".png");
101               } else {
```

```
102                 Log.e(ImageFileManager.class.toString(),
103                             "El formato del mapa de bits no es correcto");
104                 return false;
105           }
106
107           Log.i(ImageFileManager.class.toString(),
108                       "Nombre del archivo a escribir: " + fullFileName);
109
110           try {
111
112                 // Verificación de que la carpeta donde se ava a guardar el archivo
113                 // existe
114                 if (!checkSaveFolder()) {
115                       Log.e(ImageFileManager.class.toString(),
116                                   "Ha ocurrido un error con la carpeta donde se debe guardar
                                          la imagen");
117                       return false;
118                 }
119
120                 // Almacenamiento del archivo
121                 outFile = new FileOutputStream(fullFileName);
122                 bitmap.compress(bitmapFormat, 100, outFile);
123
124                 outFile.close();
125
126                 return true;
127
128           } catch (Exception e) {
129                 Log.e(ImageFileManager.class.getName(),
130                             "No ha sido posible almacenar la imagen [" + fullFileName
131                                         + "] debido a una excepción");
132                 return false;
133           }
134
135     }
136
137     /**
138      * Carga el bitmap con el formato ARGB_8888 que se encuentre en la ruta
139      * indicada por el parámetro fileName
140      *
141      * @param fileName
142      * Ruta al bitmap que se desea cargar o nombre de la imagen en
143      * caso de que se encuentre dentro de la carpeta por defecto de
144      * la aplicación
145      * @param delete
146      * Indicativo si se desea eliminar la imagen después de
147      * cargarla o no
148      * @return El bitmap cargado desde la ruta indicada en el parámetro
149      * fileName. En caso insatisfactorio se devolverá un puntero a nulo
150      */
151     public static Bitmap loadBitmapARGB_8888(String fileName, boolean delete) {
152
153           Bitmap bitmap = null;
154
```

```
155             if (!fileName.startsWith("/")) {
156                   fileName = baseFolderPath.concat(fileName);
157             }
158
159             File file = new File(fileName);
160
161             Log.i(ImageFileManager.class.toString() + ".onCreate()",
162                         "Nombre de la imagen a cargar: " + file.getAbsolutePath());
163
164             if (file.exists()) {
165                   Log.i(ImageFileManager.class.toString(),
166                               "El archivo " + file.getAbsolutePath()
167                                           + " existe y va a ser cargado");
168                   BitmapFactory.Options options = new BitmapFactory.Options();
169                   options.inPreferredConfig = Bitmap.Config.ARGB_8888;
170                   bitmap = BitmapFactory.decodeFile(file.getAbsolutePath(), options);
171
172                   if (delete) {
173                         file.delete();
174                   }
175
176                   Log.i(ImageFileManager.class.toString(),
177                               "CARGANDO IMAGEN, PROPIEDADES");
178                   Log.i("RUTA DE LA IMAGEN:", file.getAbsolutePath());
179                   Log.i("ANCHO x ALTO",
180                               bitmap.getWidth() + " x " + bitmap.getHeight());
181                   Log.i("CONFIGURACION IMAGEN:", bitmap.getConfig() + "");
182
183             } else {
184                   Log.e(ImageFileManager.class.toString(),
185                               "El archivo " + file.getAbsolutePath() + " NO EXISTE");
186             }
187
188             return bitmap;
189
190       }
191
192       /**
193        * Carga el bitmap con el formato ARGB_8888 que se encuentre en el objeto
194        * File pasado
195        *
196        * @param file
197        * - Objeto File de referencia al bitmap que se desea cargar
198        * @param delete
199        * - Indicativo si se desea eliminar la imagen después de
200        * cargarla o no
201        * @return El bitmap cargado desde la referencia indicada en el objeto file.
202        * En caso insatisfactorio se devolverá un puntero a nulo
203        */
204       public static Bitmap loadBitmapARGB_8888(File file, boolean delete) {
205
206             Bitmap bitmap = null;
207
208             if (file.exists()) {
```

```
209                 Log.i(ImageFileManager.class.toString(),
210                             "El archivo " + file.getAbsolutePath()
211                                         + " existe y va a ser cargado");
212                 BitmapFactory.Options options = new BitmapFactory.Options();
213                 options.inPreferredConfig = Bitmap.Config.ARGB_8888;
214                 bitmap = BitmapFactory.decodeFile(file.getAbsolutePath(), options);
215
216                 if (delete) {
217                       file.delete();
218                 }
219
220                 Log.i(ImageFileManager.class.toString(),
221                             "CARGANDO IMAGEN, PROPIEDADES");
222                 Log.i("RUTA DE LA IMAGEN:", file.getAbsolutePath());
223                 Log.i("ANCHO x ALTO",
224                             bitmap.getWidth() + " x " + bitmap.getHeight());
225                 Log.i("CONFIGURACION IMAGEN:", bitmap.getConfig() + "");
226
227           } else {
228                 Log.e(ImageFileManager.class.toString(),
229                             "El archivo " + file.getAbsolutePath() + " NO EXISTE");
230           }
231
232           return bitmap;
233
234     }
235 }
```

D.3. Mapeadores de la vista

Listing D.17: **aboutAppDialog.xml**

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <ScrollView >
3     <LinearLayout
4        xmlns:android="http://schemas.android.com/apk/res/android"
5        android:id="@+id/about_app_dialog"
6        android:layout_width="fill_parent"
7        android:layout_height="fill_parent"
8        android:orientation="vertical"
9        android:padding="3dp" >
10
11       <TextView
12          android:id="@+id/about_app_message"
13          android:layout_width="match_parent"
14          android:layout_height="wrap_content"
15          android:layout_margin="3dp"
16          android:text="@string/about_app_msg" />
17
18       <TextView
19          android:id="@+id/author_contact_message"
20          android:layout_width="match_parent"
21          android:layout_height="wrap_content"
```

```
22          android:layout_margin="3dp"
23          android:text="@string/author_contact_msg" />
24
25      <TextView
26          android:id="@+id/author_mail_address"
27          android:layout_width="match_parent"
28          android:layout_height="wrap_content"
29          android:layout_margin="3dp"
30          android:gravity="center"
31          android:text="@string/author_mail_address" />
32  </LinearLayout>
33  </ScrollView>
```

Listing D.18: **anisotropicDiffusion.xml**

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent" >
5
6      <ImageView
7         android:id="@+id/anisotropic_diffusion_background"
8         android:layout_width="match_parent"
9         android:layout_height="match_parent"
10        android:contentDescription="@string/noisy_image" />
11
12  </FrameLayout>
```

Listing D.19: **anisotropicDiffusionHelpDialog.xml**

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <ScrollView>
3
4      <LinearLayout
5         xmlns:android="http://schemas.android.com/apk/res/android"
6         android:layout_width="match_parent"
7         android:layout_height="match_parent"
8         android:orientation="vertical" >
9
10        <TextView
11           android:layout_width="fill_parent"
12           android:layout_height="wrap_content"
13           android:padding="3dp"
14           android:text="@string/main_anisoDiff_help_title" />
15
16        <TextView
17           android:layout_width="fill_parent"
18           android:layout_height="fill_parent"
19           android:padding="3dp"
20           android:text="@string/main_anisoDiff_help_message" />
21     </LinearLayout>
22
23  </ScrollView>
```

Listing D.20: **anisotropicDiffusionParamsDialog.xml**

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <ScrollView>
```

```
3
4   <LinearLayout
5       xmlns:android="http://schemas.android.com/apk/res/android"
6       android:layout_width="match_parent"
7       android:layout_height="match_parent"
8       android:orientation="vertical" >
9
10      <TextView
11          android:layout_width="fill_parent"
12          android:layout_height="wrap_content"
13          android:padding="5dp"
14          android:text="@string/anisotropic_diffusion_coefficient" />
15
16      <TextView
17          android:id="@+id/anisotropic_diffusion_coefficient_value"
18          android:layout_width="fill_parent"
19          android:layout_height="wrap_content"
20          android:gravity="center"
21          android:padding="5dp" />
22
23      <SeekBar
24          android:id="@+id/anisotropic_diffusion_coefficient_adjust_bar"
25          android:layout_width="match_parent"
26          android:layout_height="wrap_content"
27          android:padding="5dp" />
28
29      <TextView
30          android:layout_width="fill_parent"
31          android:layout_height="wrap_content"
32          android:padding="5dp"
33          android:text="@string/anisotropic_diffusion_time_step" />
34
35      <TextView
36          android:id="@+id/anisotropic_diffusion_time_step_value"
37          android:layout_width="fill_parent"
38          android:layout_height="wrap_content"
39          android:gravity="center"
40          android:padding="5dp" />
41
42      <SeekBar
43          android:id="@+id/anisotropic_diffusion_time_step_adjust_bar"
44          android:layout_width="match_parent"
45          android:layout_height="wrap_content"
46          android:padding="5dp" />
47
48      <TextView
49          android:layout_width="fill_parent"
50          android:layout_height="wrap_content"
51          android:padding="5dp"
52          android:text="@string/anisotropic_diffusion_num_iters" />
53
54      <EditText
55          android:id="@+id/anisotropic_diffusion_num_iters_value"
56          android:layout_width="match_parent"
```

```
         android:layout_height="wrap_content"
         android:ems="10"
         android:inputType="number"
         android:padding="5dp" />

      <TextView
         android:id="@+id/anisotropic_diffusion_params_error"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:padding="5dp"
         android:textColor="@color/rojo" />
   </LinearLayout>

</ScrollView>
```

Listing D.21: **exitDialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView>

   <LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical" >

      <TextView
         android:layout_width="fill_parent"
         android:layout_height="fill_parent"
         android:layout_gravity="center"
         android:padding="5dp"
         android:text="@string/app_exit_msg" />
   </LinearLayout>

</ScrollView>
```

Listing D.22: **explorerMain.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >

   <TextView
      android:id="@+id/path"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="left"
      android:padding="3dp"
      android:textSize="4mm"
      android:textColor="@color/rojo" />

   <ListView
      android:id="@android:id/list"
      android:layout_width="fill_parent"
```

```
      android:layout_height="wrap_content" />

   <TextView
      android:id="@android:id/empty"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/no_data" />

</LinearLayout>
```

Listing D.23: **explorerRow.xml**

```
<?xml version="1.0" encoding="utf-8"?>

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/row_text"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:padding="3dp"
   android:textSize="3mm" />
```

Listing D.24: **fileBrowserHelpDialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView>

   <LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical" >

      <TextView
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:padding="3dp"
         android:text="@string/file_browser_help_title" />

      <TextView
         android:layout_width="fill_parent"
         android:layout_height="fill_parent"
         android:padding="3dp"
         android:text="@string/file_browser_help_message" />
   </LinearLayout>

</ScrollView>
```

Listing D.25: **imageHelpDialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView>

   <LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical" >

```

```
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:padding="3dp"
        android:text="@string/image_help_title" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="3dp"
        android:text="@string/image_help_message" />
  </LinearLayout>

</ScrollView>
```

Listing D.26: **imageLoadDialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/image_load_dialog"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/file_name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:padding="5dp"
    android:text="@string/load_image_question" />

  <ImageView
    android:id="@+id/preview_image_dialog"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:contentDescription="@string/image_preview"
    android:padding="10dp" />

</LinearLayout>
```

Listing D.27: **inpaintColorHelpDialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView>

  <LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >

    <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
```

```
        android:padding="3dp"
        android:text="@string/main_inpaintColor_help_title" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="3dp"
        android:text="@string/main_inpaintColor_help_message" />
</LinearLayout>

</ScrollView>
```

Listing D.28: **inpaintColorSelection.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <com.imageRestoration.inpainting.ColorMaskSelectionTouchView
        android:id="@+id/inpaint_color_selection_background"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

    <com.imageRestoration.inpainting.ColorSelectionMaskView
        android:id="@+id/inpaint_color_mask_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

Listing D.29: **inpaintMaskHelpDialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView>

    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:padding="3dp"
            android:text="@string/main_inpaintManual_help_title" />

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="3dp"
            android:text="@string/main_inpaintManual_help_message" />
    </LinearLayout>

</ScrollView>
```

Listing D.30: **inpaintMaskSelection.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >

   <ImageView
      android:id="@+id/inpaint_manual_selection_background"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:contentDescription="@string/inpaint_mask_selection" />

   <com.imageRestoration.inpainting.ManualMaskTouchView
      android:id="@+id/inpaint_manual_mask_view"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />

</FrameLayout>
```

Listing D.31: **mainScreen.xml**

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/layout_main_screen_root"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >

   <TextView
      android:id="@+id/app_name_text"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="10"
      android:text="@string/app_name"
      android:textColor="@color/verde"
      android:textSize="30sp"
      android:textStyle="bold"
      android:gravity="center" />

   <LinearLayout
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:orientation="horizontal"
      android:layout_gravity="bottom"
      android:gravity="center" >

      <Button
         android:id="@+id/open_file_button"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/open_file" />

      <Button
         android:id="@+id/about_app_button"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/about_app" />
    </LinearLayout>

</LinearLayout>
```

Listing D.32: **mainScreenHelpDialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView>

    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:padding="3dp"
            android:text="@string/main_screen_help_title" />

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:padding="3dp"
            android:text="@string/main_screen_help_message" />
    </LinearLayout>

</ScrollView>
```

Listing D.33: **saveImageDialog.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView>

    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:padding="5dp"
            android:text="@string/save_image_msg" />

        <TextView
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:padding="5dp"
            android:text="@string/save_image_name" />

```

```
22      <EditText
23         android:id="@+id/new_file_name"
24         android:layout_width="fill_parent"
25         android:layout_height="wrap_content"
26         android:inputType="text"
27         android:padding="5dp" />
28
29      <TextView
30         android:layout_width="fill_parent"
31         android:layout_height="wrap_content"
32         android:padding="5dp"
33         android:text="@string/save_image_type" />
34
35      <Spinner
36         android:id="@+id/image_type_spinner"
37         android:layout_width="fill_parent"
38         android:layout_height="wrap_content"
39         android:padding="5dp"
40         android:prompt="@string/save_image_select_type" />
41   </LinearLayout>
42
43 </ScrollView>
```

Listing D.34: **showImage.xml**

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent" >
5
6     <ImageView
7        android:id="@+id/prerestored_image"
8        android:layout_width="match_parent"
9        android:layout_height="match_parent"
10       android:contentDescription="@string/prerestored_image"
11       android:padding="5dp" />
12
13 </FrameLayout>
```

D.4. Mapeadores de los menús

Listing D.35: **anisotropicDiffusionMenu.xml**

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <menu xmlns:android="http://schemas.android.com/apk/res/android" >
3
4     <item
5        android:id="@+id/anisotropic_diffusion_accept_opt"
6        android:title="@string/set_params"/>
7     <item
8        android:id="@+id/anisotropic_diffusion_help_opt"
9        android:icon="@android:drawable/ic_menu_help"
10       android:title="@string/help"/>
11    <item
```

```
      android:id="@+id/anisotropic_diffusion_back_opt"
      android:icon="@android:drawable/ic_menu_revert"
      android:title="@string/back"/>

</menu>
```

Listing D.36: **fileBrowserMenu.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

   <item
      android:id="@+id/file_browser_help_opt"
      android:icon="@android:drawable/ic_menu_help"
      android:title="@string/help"/>
   <item
      android:id="@+id/file_browser_back_opt"
      android:icon="@android:drawable/ic_menu_revert"
      android:title="@string/file_browser_back"/>

</menu>
```

Listing D.37: **imagePreviewMenu.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

   <item
      android:id="@+id/inpaint_opt"
      android:title="@string/inpaint"/>
   <item
      android:id="@+id/denoise_opt"
      android:title="@string/denoise"/>
   <!-- item
      android:id="@+id/deblur_opt"
      android:title="@string/deblur"/-->
   <item
      android:id="@+id/save_opt"
      android:icon="@android:drawable/ic_menu_save"
      android:title="@string/save"/>
   <item
      android:id="@+id/image_help_opt"
      android:icon="@android:drawable/ic_menu_help"
      android:title="@string/help"/>
   <item
      android:id="@+id/load_other_image_opt"
      android:icon="@android:drawable/ic_menu_gallery"
      android:title="@string/load_other_image"/>

</menu>
```

Listing D.38: **inpaintingSelectionMenu.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

   <item
      android:id="@+id/inpainting_accept_opt"
```

```
      android:title="@string/accept"/>
   <item
      android:id="@+id/inpainting_reset_opt"
      android:title="@string/reset_selection"/>
   <item
      android:id="@+id/inpainting_help_opt"
      android:icon="@android:drawable/ic_menu_help"
      android:title="@string/help"/>
   <item
      android:id="@+id/inpainting_back_opt"
      android:icon="@android:drawable/ic_menu_revert"
      android:title="@string/file_browser_back"/>

</menu>
```

Listing D.39: **mainScreenMenu.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

   <item
      android:id="@+id/general_help_opt"
      android:icon="@android:drawable/ic_menu_help"
      android:title="@string/help"/>
   <item
      android:id="@+id/exit_opt"
      android:icon="@android:drawable/ic_menu_close_clear_cancel"
      android:title="@string/app_exit"/>

</menu>
```

D.5. Contenido de los campos estáticos

Listing D.40: **strings.xml**

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<resources>
   <!-- Cadenas de texto de propósito general para la aplicación -->
   <string name="app_name">Photo Restore</string>
   <string name="accept">Aceptar</string>
   <string name="cancel">Cancelar</string>
   <string name="help">Ayuda</string>
   <string name="back">Volver</string>
   <string name="app_exit">Salir</string>
   <string name="app_exit_msg">¿Está seguro de que desea salir de la aplicación?</string>
   <!-- Campos sobre la ventana de diálogo sobre la aplicación -->
   <string name="open_file">Abrir archivo</string>
   <string name="about_app">Acerca de...</string>
   <string name="about_app_msg">Esta aplicación fue desarrollada por Samuel Martinez como
        Tesina del Máster en Modelización Matemática y Computación en la Universidad de
        Valladolid.</string>
   <string name="author_contact_msg">Contacte con el autor ante cualquier percance, fallo o
        duda:</string>
   <string name="author_mail_address">mo.samuel@gmail.com</string>
```

```
17  <!-- Campos sobre la ventana de exploración de ficheros -->
18  <string name="file_browser">Explorador de ficheros</string>
19  <string name="current_location">Localización actual: </string>
20  <string name="sdcard_root">Carpeta raíz tarjeta SD</string>
21  <string name="file_icon">Icono del archivo</string>
22  <string name="file_name">Nombre del archivo</string>
23  <string name="file_browser_back_folder">../</string>
24  <string name="file_browser_back">Volver</string>
25  <string name="no_data">No existen datos que mostrar</string>
26  <!-- Campos sobre la ventana de diálogo con la previsualización de la imagen -->
27  <string name="image_preview">Previsualización de la imagen</string>
28  <string name="load_image_question">¿Desea usted cargar esta imagen?</string>
29  <!-- Campos de la ventana de diálogo que se mostrará si el fichero no es una imagen -->
30  <string name="not_an_image_msg">El fichero seleccionado no es una imagen o no tiene un
        formato aceptado por la aplicación</string>
31  <!-- Campos de la ventana de diálogo que se mostrará si la carpeta a visualizar no tiene
        permisos de lectura -->
32  <string name="not_readable_folder_msg">¡La carpeta no se puede leer!</string>
33  <!-- Campos para mostrar la imagen y las opciones de restauración en el menú -->
34  <string name="prerestored_image">Imagen antes de restaurar</string>
35  <string name="inpaint">Inpaint</string>
36  <string name="denoise">Reducir ruido</string>
37  <string name="deblur">Enfocar</string>
38  <string name="save">Guardar</string>
39  <string name="load_other_image">Cargar otra imagen</string>
40  <string name="load_other_image_dialog_msg">¿Desea cargar otra imagen?</string>
41  <!-- Campos a mostrar en la acción de inpainting -->
42  <string name="inpaint_type_mask_selection_question">¿Qué forma de selección máscara desea
        utilizar?</string>
43  <string name="inpaint_color_selection">Mediante color</string>
44  <string name="inpaint_mask_selection">Manual</string>
45  <string name="inpaint_user_select_color">Seleccione el color dentro del cual se va a
        realizar el Inpainting</string>
46  <string name="inpaint_user_select_mask">Seleccione la máscara dentro de la que realizar el
        Inpainting</string>
47  <string name="reset_selection">Reiniciar selección</string>
48  <!-- Campos para la ventana de diálogo de guardar la imagen -->
49  <string name="save_image_msg">El fichero será almacenado en la carpeta \\sdcard\\
        PhotoRestore\\</string>
50  <string name="save_image_name">Nombre del archivo:</string>
51  <string name="save_image_type">Tipo del archivo:</string>
52  <string name="save_image_select_type">Seleccione un tipo de archivo</string>
53  <string name="save_image_success">La imagen se ha guardado correctamente</string>
54  <string name="save_image_fail">La imagen no se ha podido almacenar</string>
55  <string-array name="save_image_types">
56     <item>PNG</item>
57     <item>JPEG</item>
58  </string-array>
59  <string name="save_image_file_name_mistake">Se ha introducido dentro del nombre del fichero
         algún caracter no aceptado.\n Los caracteres no aceptados son\n , . \\ / | # @</
        string>
60  <string name="save_image_file_type_mistake">Para continuar debe seleccionar un tipo de
        archivo</string>
61  <!-- Campos te de texto para el módulo de difusión anisótropa -->
```

```
62    <string name="noisy_image">Imagen ruidosa</string>
63    <string name="set_params">Establecer parámetros</string>
64    <string name="anisotropic_diffusion_coefficient">Coeficiente de difusividad</string>
65    <string name="anisotropic_diffusion_time_step">Paso temporal</string>
66    <string name="anisotropic_diffusion_num_iters">Número de iteraciones</string>
67    <string name="anisotropic_diffusion_error_param_num_iters">Número de iteraciones incorrecto
          </string>
68    <string name="anisotropic_diffusion_error_param_time_step">Valor del paso temporal
          incorrecto</string>
69    <string name="anisotropic_diffusion_error_param_k">Valor del coeficiente de difusividad
          incorrecto</string>
70    <!-- Campos para mostrar en las ventanas de diálogo de la ayuda -->
71    <string name="help_message_default">Texto ventana de ayuda</string>
72 </resources>
```

Listing D.41: **helpStrings.xml**

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4     <!-- Ayuda de la ventana principal -->
5     <string name="main_screen_help_title">Ayuda de la pantalla principal</string>
6     <string name="main_screen_help_message">
7        El fundamento de la aplicación está basado en la restauración de imágenes digitales.
8        Para comenzar al restauración debe elegir una de las imágenes almacenadas en su
             dispositivo mediante el botón Abrir Archivo</string>
9
10    <!-- Ayuda del file browser -->
11    <string name="file_browser_help_title">Ayuda del explorador de ficheros</string>
12    <string name="file_browser_help_message">
13       En esta sección puede explorar la memoria de su teléfono y elegir una imagen para
             cargarla y poder comenzar con su restauración.
14    </string>
15
16    <!-- Ayuda de la imagen -->
17    <string name="image_help_title">Ayuda de la visualización de la imagen</string>
18    <string name="image_help_message">
19       En esta sección puede seleccionar varias vías posibles:
20       Inpaint: Es un método de procesado avanzado que permite eliminar elementos dentro de una
              imagen y automáticamente los rellenará con información que encuentre alrededor de
             la región que se indique. Los puntos a restaurar serán seleccionados mediante una
             máscara. Existen dos modalidades de selección una en la que el usuario dibuja sobre
              los puntos que se van a restaurar y otra donde el usuario selecciona el color que
             quiere que desaparezca.
21       Reducir ruido: Posibilita al usuario a quitar esos puntitos molestos que se producen
             cuando se toman fotografías.
22       Guardar: Abre una ventana para que se seleccione el nombre del fichero a guardar y su
             formato (a elegir entre JPG y PNG). Las imágenes serán guardadas dentro de la ruta
             /sdcard/PhotoRestore/
23       Cargar otra imagen: Devuelve al usuario al explorador de ficheros para cargar un nuevo
             fichero.</string>
24
25    <!-- Ayuda de la ventana de difusión anisótropa -->
26    <string name="main_anisoDiff_help_title">Ayuda para la reducción de ruido</string>
27    <string name="main_anisoDiff_help_message">
```

```
28        Para la reducción de ruido se utiliza una técnica denominada filtrado de difusión
              anisótropa.
29        Necesita tres parámetros para funcionar correctamente, los cuales debe seleccionar:
30        Coeficiente de difusividad: Este parámetro hace que el algoritmo distinga más o menos
              los bordes. Cuanto menor sea, más bordes distinguirá. Si es muy grande, pasará por
              alto la mayoría de los detalles.
31             Paso temporal: Se selecciona lo rápido que se quiere que avance el algoritmo en
                   cada paso que dé.
32             Número de iteraciones: El número de pasos que se va a ejecutar el algoritmo.
                   Cuanto mayor sea, más tardará, pero más ruido habrá quitado. Si este número
                   es muy grande pueden desaparecer demasiados detalles en la imagen.</string>
33
34    <!-- Ayuda de la ventana de inpainting selección manual -->
35    <string name="main_inpaintManual_help_title">Ayuda para el inpainting manual</string>
36    <string name="main_inpaintManual_help_message">
37       Este es el método para la restauración mediante la selección manual.
38       Debe desplazar el dedo sobre la imagen para pintar la zona que se desea restaurar.
39       Esta selección se puede reiniciar en cualquier momento. Un modo de reinicio es el cambio
              de orientación de la pantalla si tiene activado el acelerómetro.</string>
40
41    <!-- Ayuda de la ventana de inpainting selección por color -->
42    <string name="main_inpaintColor_help_title">Ayuda para el inpainting seleccionando color</
          string>
43    <string name="main_inpaintColor_help_message">
44       Este es el método para la restauración mediante la selección mediante color.
45       Debe pulsar con el dedo sobre la imagen para seleccionar el color que se desea restaurar
              . Se pueden seleccionar varios colores.
46       Esta selección se puede reiniciar en cualquier momento. Un modo de reinicio es el cambio
              de orientación de la pantalla si tiene activado el acelerómetro.</string>
47
48 </resources>
```

Listing D.42: **colors.xml**

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3     <color name="rojo">#FF0000</color>
4     <color name="verde">#00FF00</color>
5     <color name="azul">#0000FF</color>
6  </resources>
```

D.6. Configuración de la aplicación

Listing D.43: **AndroidManifest.xml**

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="com.imageRestoration"
4     android:versionCode="1"
5     android:versionName="1.0" >
6
7     <uses-sdk
8        android:minSdkVersion="8"
9        android:targetSdkVersion="8" />
```

```
10
11    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
12
13    <application
14       android:icon="@drawable/ic_launcher"
15       android:label="@string/app_name" >
16       <activity
17          android:name=".mainScreen.MainScreenActivity"
18          android:label="@string/app_name" >
19          <intent-filter>
20             <action android:name="android.intent.action.MAIN" />
21
22             <category android:name="android.intent.category.LAUNCHER" />
23          </intent-filter>
24       </activity>
25       <activity
26          android:name=".fileBrowser.FileBrowserActivity"
27          android:description="@string/file_browser"
28          android:label="@string/app_name" >
29       </activity>
30       <activity
31          android:name=".showImage.ShowImageActivity"
32          android:configChanges="keyboardHidden|orientation"
33          android:description="@string/image_preview"
34          android:label="@string/app_name" >
35       </activity>
36       <activity
37          android:name=".inpainting.ImageInpaintingColorSelectionActivity"
38          android:configChanges="keyboardHidden|orientation"
39          android:label="@string/app_name" >
40       </activity>
41       <activity
42          android:name=".inpainting.ImageInpaintingMaskSelectionActivity"
43          android:configChanges="keyboardHidden|orientation"
44          android:label="@string/app_name" >
45       </activity>
46       <activity
47          android:name=".anisotropicDiffusion.AnisotropicDiffusionActivity"
48          android:configChanges="keyboardHidden|orientation"
49          android:label="@string/app_name" >
50       </activity>
51    </application>
52
53 </manifest>
```